UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE FÍSICA

Cosmological Density Fields and Gravitational Waves: A Statistical Learning Approach

Miguel João Pires Carreira da Conceição

Mestrado em Física
Especialização em Astrofísica e Cosmologia

Dissertação orientada por:
Dr. António José Cunha da Silva
Dr. Alberto Garcez de Oliveira Krone Martins

2021

I dedicate this thesis to my parents. They may have parted, but will always be present with each of my steps.

Acknowledgments

As Bernard of Chartres said in the 12th century, and later Isaac Newton re-mentioned, we “Stand on the Shoulders of Giants”.

I should begin by acknowledging all the previous scientists, which provided an incredibly large library of thoughts and tools, enabling us to reach the high-level of nature’s description and manipulation we see today. This work stands on hundreds of thousands of years of *Homo-Sapiens* understanding of nature’s rules. So it is reasonable to give credit to all these generations.

I thank my supervisors, my most modern and closest “Giants”, for all the time, knowledge and good moments they provided me, as well as the patience in guiding a new-born in the field. I thank my friends for putting up with my possibly excessive conversations on the topics of this thesis. I thank my sisters for putting up with my possibly over-stressed tantrums. Finally, I thank the all living science community for never giving up on the quest for increasing knowledge.

May the ride never end...

Abstract

Several cosmological and astrophysical studies rely on the generation of costly simulations to sample the parameter space of an underlying physical theory. Historically, the study of gravity and gravitational effects has been among the most important examples of such cases, and gravity has been linked to the very requirements to build the largest available computational facilities. This thesis concerns the construction of methods to alleviate these requirements, and to enable significantly faster sampling and inference. To do so, we adopted supervised machine learning algorithms coupled with a basis transformation method, and we applied our methods to two cosmological and astrophysical scenarios where gravity dominates at different scales and strength regimes.

In a first scenario, we study the emulation of 3D N-body simulations of the dark matter density field. We construct a method that uses different machine learning methods and adopts Principal Component Analysis to enable fast emulations, and we study how it behaves in two application cases. In the first case we perform the emulation considering a single free-parameter, Ω_{dm}, the dark matter density. In the second case, we perform the emulation considering two free-parameters, Ω_{dm} and redshift. We additionally demonstrate an application of *Functional Principal Component Analysis* (FPCA) on the single free-parameter case, which is probably the first-ever application of this method in a cosmological context. Although we applied our method to dark matter density fields, it is generic enough to emulate any other 3D scalar field.

In a second scenario, we study the application of the proposed methodology to perform parameter inference of progenitors of gravitational waves. We adapt our method to learn from approximated simulations of gravitational waveforms the value of the combination of the masses of the progenitor system, the chirp mass parameter, M_{chirp}. We demonstrate the application of our method in two inference cases: in time domain and a more realistic frequency domain case, and in both cases, we consider realistic noise distributions from ground-based gravitational wave detectors.

We show that our proposed approach enables gains of three orders of magnitude in running times compared to performing a full N-body simulation, while still reproducing the power spectrum and bispectrum within $\sim 1\%$ and $\sim 3\%$ errors for the single parameter emulation and $\sim 5\%$ and $\sim 15\%$ for the two-parameter emulation. We also show that in a fraction of a second, our method allows the inference of the chirp masses from gravitational wave data with errors within $\sim 2\%$ in the high signal-to-noise ratio regime and $\sim 5\%$ in the low signal-to-noise regime. These results indicate that the proposed methodologies are promising alternatives to analyse the large datasets of cosmological and astrophysical observations expected to result from upcoming surveys like Euclid, LSST, and LISA, and thus to help in constraining the models that describe the behaviour of the most important interaction that shapes our vision of the Universe: Gravity.

Keywords: N-body Simulations, Structure Formation, Gravitational Waves, Machine Learning, Principal Component Analysis.

Resumo

Vários estudos em cosmologia e astrofísica dependem da geração de simulações de alta performance para explorar o espaço de parâmetros que rege as teorias físicas subjacentes. Historicamente, o estudo da gravidade e efeitos gravitacionais representa um dos mais importantes exemplos dessa dependência, sendo que os recursos computacionais requeridos para simular a dinâmica gravitacional têm contribuido para a necessidade de construção das maiores infra-estruturas computacionais modernas.

Esta tese tem como tema a construção de métodos para aliviar esses recursos, diminuindo o tempo necessário para a exploração e inferência no espaço de parâmetros. Para o efeito, adoptamos quatro algoritmos supervisionados de Aprendizagem Automática (*Machine Learning*), Redes Neuronais (*Neural Networks*), Floresta Aleatória (*Random Forest*), Árvores Extremamente Aleatórias (*Extremelly Randomized Trees*) e Máquinas de Vectores de Suporte (*Support Vector Machines*), em conjunto com um método de tranformação de base (Análise de Componentes Principais), aplicando-os a dois cenários astrofísicos e cosmológicos onde a gravidade é dominante, a escalas e magnitudes opostas.

No primeiro cenário, estudamos a emulação de campos de densidade de matéria escura com base em simulações de N-corpos. Neste caso, o nosso método usa os diferentes algoritmos de *Machine Learning* supervisionado e adopta "Análise de Componentes Principais" para permitir a realização de emulações rápidas, e estudamos a sua aplicação em dois casos. No primeiro, realizamos emulações considerando apenas um parâmetro livre, Ω_{dm}, a densidade de matéria escura. No segundo caso, de forma a testar os nossos algoritmos em contextos de maior dimensionalidade, consideramos dois parâmetros livres, Ω_{dm} e o *redshift* (z).

Partindo de um conjunto de simulações de N-corpos, que serve como "exemplo de treino" os algoritmos conseguem aprender a relação entre os dados das simulações e um dado conjunto de parâmetros livres de interesse. De forma a tornar o problema o mais computacionalmente tratável possível, aplicamos Análise de Componentes Principais ao conjunto de dados, previamente à aprendizagem. Este método toma partido da redundância intrínseca aos dados, projetando-os numa base que preserva o máximo da sua variância num conjunto mínimo de coeficientes, chamados "Componentes Principais". Posteriormente à projeção, segue-se a aprendizagem. Em vez de fornecer os dados das simulações diretamente aos algoritmos de *Machine Learning*, são os Componentes Principais dos dados projetados que são fornecidos, permitindo uma redução nos recursos computacionais necessários. Neste cenário em particular, usamos os Componentes Principais como variáveis dependentes e os parâmetros cosmológicos como variáveis independentes. Posteriormente à aprendizagem, com base num novo conjunto de parâmetros cosmológicos de teste, é possível estimar os Componentes Principais relativos à simulação correspondente. Estes novos Componente Principais podem então ser projetados na base original, o campo de densidades, que corresponderá à emulação final.

Como medidas de comparação das nossas emulações com as simulações reais, usamos uma medida de comparação de magnitudes de densidade célula a célula, a que chamamos de *Mean Over-density Distance*, e duas medidas de comparação de propriedades estatísticas, o *Power Spectrum* e o *Bispectrum*, sendo as duas últimas as nossas métricas príncipais. O *Power Spectrum* corresponde à transformada de Fourier da função de correlação a dois pontos, e fornece uma descrição completa de um campo aleatório Gaussiano, enquanto que o *Bispectrum* corresponde à transformada de Fourier da função de correlação a três pontos e analisa desvios à Gaussianidade da distribuição. Consequentemente, juntas, estas quantidades fornecem

uma descrição consideravelmente alargada das propriedades estatísticas dos nossos campos de densidade.

Adicionalmente, realizamos uma demonstração da aplicação de "Análise de Componentes Principais Funcional" no caso de um parâmetro livre, sendo esta provavelmente a primeira aplicação do método mencionado num contexto cosmológico. Este método transforma os nossos dados de treino em formato funcional (continuo), providenciando um conjunto de funções continuas que relacionam cada Componente Principal com o parâmetro cosmológico de interesse, permitindo assim a realização das emulações dos campos de densidade sem ser necessária a introdução de algoritmos de aprendizagem supervisionada, e consequentemente, a elaboração de uma metodologia igualmente útil, mas mais compacta. Adicionalmente, quando comparada com a metodologia anterior, no mesmo conjunto de dados, este método permite a obtenção de erros inferiores e resultados mais robustos.

Um aspeto importante de se sublinhar consiste no facto de apesar destas aplicações serem especificas a campos de densidade de matéria escura, o nosso método é genérico o suficiente para permitir a emulação de qualquer campo escalar em 3D, como por exemplo o campo de densidades de matéria bariónica.

Num segundo cenário, estudamos a aplicação da metodologia proposta para realizar inferência de parâmetros relativos aos progenitores de ondas gravitacionais. Os nossos métodos são adaptados de forma a aprender a relação entre os vetores de amplitudes de simulações aproximadas de ondas gravitacionais emitidas por sistemas binários de Buracos Negros e o *Chirp Mass* (M_{chirp}) do sistema que emite a onda, que corresponde a uma quantidade que contém o valor das massas de ambos os corpos.

Demonstramos a aplicação deste método em dois casos: domínio temporal e de frequências, e em ambos os casos, consideramos distribuições de ruído realisticas, esperadas em detetores terrestres de ondas gravitacionais.

Relativamente a este cenário, aplicamos o mesmo método de decomposição do conjunto de dados de ondas gravitacionais em Componentes Principais, seguindo-se a aplicação dos algoritmos de aprendizagem supervisionada. No entanto, uma vez que o objetivo neste caso consiste em prever o parâmetro livre e não a onda em si, os Componentes Principais terão o papel de variáveis independentes. Posteriormente à aprendizagem, partindo de um dado conjunto de ondas de teste não incluído no conjunto de treino, é possivel projetá-lo na base de Componentes Principais do conjunto de treino. Estes novos componentes principais, podem então ser fornecidos aos algoritmos treinados para inferência do *Chirp Mass* da onda correspondente. De forma a testar os nossos métodos em diferentes regimes de razão sinal-ruído calculamos o ruído esperado no detetor terrestre *Advanced LIGO* e adicionamo-lo a conjuntos de ondas gravitacionais emitidas por sistemas binários a distâncias gradualmente superiores. Desta forma cada conjunto de ondas terá uma razão sinal-ruído gradualmente inferior.

Neste caso, uma vez que as nossas grandezas estimadas (*Chirp Masses*) correspondem a grandezas de natureza contínua, decidimos aplicar como estatistica o Desvio Quadrático Medio (*Root Mean Squared Error*) para comparar as nossas estimações com os parâmetros reais.

Mostramos que a nossa abordagem proposta permite um ganho de três ordens de magnitude na obtenção dos campos de densidade, comparativamente ao tempo de execução de uma simulação de N-corpos da mesma escala e no mesmo sistema. Adicionalmente, conseguimos reproduzir o *Power Spectrum* e *Bispectrum* com erros abaixo de $\sim 1\%$ e $\sim 3\%$ para o caso de emulação a partir de um parâmetro livre (Ω_{dm}) e abaixo de $\sim 5\%$ e $\sim 15\%$ para o caso de emulação a partir de dois parâmetros livres (Ω_{dm}, z). Sendo que os algoritmos com maior prestação foram as *Neural Networks* no caso de emulação a partir de um parâmetro livre e as *Support Vector Machines* no caso de emulação a partir de dois parâmetros livres. No segundo cenário, mostramos que numa fração de segundo, o nosso método permite a inferência de 255 *Chirp Masses* a partir de dados de ondas gravitacionais com erros abaixo dos $\sim 2\%$ em regimes de elevada razão sinal-ruído e usando o algoritmo *Extremelly Randomized Trees*, e abaixo dos $\sim 5\%$

em regimes de baixa razão sinal-ruído e usando o algoritmo *Support Vector Machines*. Estes resultados indicam que as metodologias propostas consistem em alternativas promissoras para a análise de grandes conjuntos de dados provenientes de observações cosmológicas e astrofísicas que serão brevemente realizadas por missões como *Euclid*, *LSST*, e *LISA*. Ajudando assim a constranger os modelos que descrevem o comportamento da interação mais importante que molda a nossa visão do Universo: A Gravidade.

Palavras-chave: Simulações de N-corpos, Formação de Estrutura, Ondas Gravitacionais, Aprendizagem Automática, Análise de Componentes Principais.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last few decades, with the massification of numerical simulations and the increase of computing power, many research areas in Astronomy have become flooded with incredible amounts of data. This has contributed to a growing awareness about the beginning of a new era of science which incorporates the need to develop appropriate tools to deal with these increasingly large data sets. The gradual entanglement of these tools into the very fabric of how we think while doing research is usually referred to as the *fourth paradigm of science* [1]. Astrophysics and Cosmology are two fields where the need to develop such tools was rapidly noticed mostly due to sky surveys, such as *Sloan Digital Sky Survey* (SDSS) that produced around $\sim$ 20 Gb of data every night due to photometric and spectroscopic observations and data acquisition for millions of objects. In particular, for cosmology in optical wavelengths, the mission *Euclid* and the *Large Synoptic Survey Telescope* (LSST) will soon provide researchers in these fields with an unprecedented amount of data, reaffirming the need to develop new tools for data compression and data mining.

Fortunately, the growth in computing power has also allowed the emergence of a fast-growing field of statistics and computing science, commonly referred to as Machine Learning (ML), that provides efficient methods to deal with *big data*, in an unprecedented way. Machine Learning methods are usually classified into the categories of supervised and unsupervised learning. The first can be used to analyze large data sets and estimate their underlying distributions, which can then be applied to different, previously unseen, data sets. The second can be used as a tool for data compression, discovery and visualization, providing means to handle large data sets with minimal computational resources, and helping to find subtle and possibly previously unnoticed patterns in the data.

This work addresses the application of supervised and unsupervised machine learning to two of the richest (and iconic) fields of Cosmology and Astrophysics: large scale structure and gravitational waves. In a way, we can see this work as one of the first applications of these modern methods in two different contexts where gravity is the dominant force but operates in opposite regimes of scale and magnitude.

Regarding large-scale structure, our goal is to generate fast and accurate 3D estimations of Dark Matter density fields for a given choice of cosmological parameters. To achieve that, we combine supervised machine learning methods with Principal Component Analysis (PCA), using the outputs of N-body simulations and their corresponding cosmological parameters as references to training the ML algorithms. We implement this in two steps. First, we use a single cosmological parameter, the dark matter density (Ω_{dm}), to build the ML regressions. Next, we introduce the redshift (z) as an additional free parameter to study the method's performance with more free parameters and to test its ability to describe structure formation over time.

For gravitational waves, our goal is to use the same machine learning approach but with a different objective. Instead of implementing these methods to emulate gravitational waveforms given a set of astrophysical source parameters, we do the opposite and apply them on previously generated waveforms to perform inferences on the parameters of the gravitational wave emitting system. To achieve that, we apply the same methodology of using supervised machine learning together with PCA on a dataset of wave-

forms generated by the LAL (Ligo Algorithms Library) Simulation [2] approximants. The astrophysical parameter chosen as a free parameter in this context is the Chirp Mass (M_{chirp}) of the Gravitational Wave (GW) emitting binary. To test the performance of the algorithms we also use LAL simulated waveforms, but to get closer to reality, we add noise to those waveforms and make the parameter inferences in these conditions, to mimic a realistic GW parameter inference scenario. As in the large scale structure case, this is done in two steps. In the first, we work with Time Domain waveforms while in the second we work with more realistic Frequency Domain waveforms. To generate both of these waveform types and corresponding noise levels we use the PyCBC platform [3, 4] that is also based on the aforementioned LAL Simulation Library.

1.1 Cosmological Density Fields

In this Section, we introduce the main concepts and mathematical framework used in observational cosmology to describe the formation and evolution of large-scale structure. Since we are dealing with dark matter density fields, evolving under the gravitational force on large scales, we will present Einstein's field equations, the Λ-CDM model, as well as the cosmological parameters, pertained in it. We will also review the statistical tools used to characterize the density field, such as the matter power spectrum and bispectrum. Finally, we will briefly overview the main numerical N-body techniques that are used in cosmology to model the non-linear evolution of cosmic structure.

1.1.1 The Standard Model of Cosmology

The Standard Model of Cosmology, Λ-*CDM*, describes the Universe as an expanding homogeneous and isotropic fluid where its gravitational dynamics are described by Einstein's field equations, given by:

$$G_{ab} = R_{ab} - \frac{1}{2} g_{ab} R = \frac{8\pi G}{c^4} T_{ab} + \Lambda g_{ab} \tag{1.1.1}$$

where G_{ab} is the Einstein tensor, G is the Gravitational Constant,c is the speed of light, R_{ab} and R are the Ricci tensor and scalar, respectively, T_{ab} is the energy-momentum tensor and Λ the cosmological constant. These equations translate the interplay between the curvature of space-time and the energy/matter distribution it contains. The space-time curvature is described by the Ricci tensor and the Ricci scalar, and the energy distribution by the energy-momentum tensor, as the name implies. One obtains Einstein's equations for a given coordinate system, by specifying the line element:

$$ds^2 = g_{ab} dx^a dx^b \tag{1.1.2}$$

where g_{ab} is called the *metric tensor*, which specifies the geometry of space-time. If one imposes the Cosmological Principle and works in reduced-circumference polar coordinates (r,θ,ϕ) the solution of (1.1.1) is the FLRW (*Friedmann–Lemaître–Robertson–Walker*) line element:

$$ds^2 = c^2 dt^2 - a^2(t) \left[\frac{dr^2}{1 - kr^2} + r^2 (d\theta^2 + sin^2(\theta) d\phi^2) \right] \tag{1.1.3}$$

where t is the Universal time, c the speed of light and a the scale factor, which accounts for the dynamics (expansion or contraction) of the spatial part of the metric. Regarding the energy-momentum tensor, the imposition of homogeneity and isotropy enables us to express it in the same way as we would for a perfect

fluid:

$$T_{ab} = \left(\rho + \frac{p}{c^2}\right)U_a U_b - \frac{p}{c^2} g_{ab} \tag{1.1.4}$$

where $\rho(t)$ and $p(t)$ are the energy density and pressure of the fluid, respectively, and U_a is the four-velocity field of the fundamental observer. Through this expression and the FLRW line element, the solution of Einstein's equations yields the so-called Friedmann equation:

$$\frac{8\pi G}{3H^2}\rho_m + \frac{\Lambda c^2}{3H^3} - \frac{kc^2}{a^2H^2} = 1 \tag{1.1.5}$$

where $H = \dot{a}/a$ is the Hubble function, Λ and k are the cosmological constant and the space curvature, respectively. The first term accounts for the matter-radiation density, including radiation and baryonic and dark matter, so we have $\rho_m = \rho_r + \rho_{dm} + \rho_{bar}$. The second term accounts for the density of dark energy, responsible for the accelerated expansion of the Universe. The Hubble function gives the expansion rate of the universe and provides a direct proportionality relation, also known as the Hubble law, between the distance, r, and the expansion recession velocities, $v_H = Hr$, of distant light-emitting objects that we observe as having a cosmological redshift, z, in their spectra. The Hubble function also enters in the definition of the critical density, $\rho_c = 3H/8\pi G$, that allows the definition of the density parameters, as discussed below. Finally, the third term accounts for the universe's curvature, which is consistent with zero according to Planck observations [5].

Dividing (1.1.5) by ρ_c we can write the Friedmann equation in a more compact form, expressing a conservation law:

$$\Omega_m + \Omega_\Lambda + \Omega_k = 1 \tag{1.1.6}$$

where $\Omega_m = \Omega_r + \Omega_{dm} + \Omega_{bar}$ and Ω_k are the density parameters of matter-radiation and curvature respectively and Ω_Λ is the density parameter of dark energy. For more details on General Relativity and Cosmology please see [6, 7]. These density parameters, the Hubble expansion rate at present, H_0 and other quantities that we will introduce in the next Section are part of the set of cosmological parameters that define a given cosmological model.

In this work, we will only be dealing with dark matter simulations in a Λ-*CDM* paradigm. So we chose to set simulations (the initial conditions of our N-body snapshots) with Ω_{dm} varying in a range of values, and with all the remaining cosmological parameters set to values consistent with present constraints from Planck observations. This means that we will be working with purelly dark matter simulations where baryons are treated as dark matter as well.

1.1.2 Structure Formation and the Matter Power Spectrum

The formation of structures such as galaxies and clusters has still many unknowns [8]. The present stance is that small density perturbations in the primordial universe gave rise to the inhomogeneous structures we see today [9]. The evolution of the perturbations in an expanding universe is well described by linear perturbation theory if we consider only the initial stages of structure formation when the perturbations were still small. This stage corresponds to the so-called *linear regime* of density perturbations.

As perturbations grow, linear theory breaks down and it becomes harder to find analytical solutions to describe their posterior evolution. Nonetheless, it is possible to follow structure formation even throughout the more advanced stages of non-linear evolution we observe today, using numerical methods such as N-body simulations [10]. These simulations consider a given set of massive particles interacting through their mutual gravity in a cubic box of fixed size. Given appropriate initial conditions, they are evolved

through time by summation of their pairwise interactions with their Newtonian force equations solved iteratively. In each time step, the resulting acceleration is used to update each particle position and velocity. Several simulation outputs (snapshots) with particle positions and velocities can be produced at any given time (redshift). The set of outputs produced in this way provides a realization run, from which one can compute the overdensity field (see below) at a set of redshifts for a given choice of cosmological parameters. Given these outputs and the appropriate observable quantities, one can compare simulations to the observed universe, to improve constraints on structure formation mechanisms and cosmological parameters of the Λ-*CDM* model [5].

The initial conditions of the N-body simulations require the computation of the amplitudes of density field at some initial redshift, in a way consistent with the primordial matter power spectrum of the cosmological model (see below). The matter power spectrum is the Fourier Transform of the two-point correlation function of the initial density fluctuation field predicted by fundamental theory (typically in the context of the inflationary theory). This quantity provides a complete statistical characterization for gaussian random fields. The most popular models of inflation [11], predict a gaussian, scale-invariant, primordial power spectrum for the density perturbations. At this stage and during the linear evolution of density perturbations the matter power spectrum fully describes the matter density field. As perturbations evolve non-linearly, non-gaussianities will develop and higher-order statistics need to be introduced to fully describe the density field in the Universe. The power spectrum of the matter density field at latter times, e.g. at the initial conditions redshift of our simulation runs, can be computed through the use of a transfer function, usually provided by Boltzmann codes, which relates the shape of the current power spectrum with its primordial form.

The matter power spectrum can be computed directly from N-body simulations (as well as from observations). Therefore it is a key statistical tool to characterize the density field in N-body simulations even through the non-linear evolution of structure formation where non-gaussianities naturally arise. It is thus often used (along with higher-order statistics like the bispectrum), to compare the density fields generated by simulations with the observed ones, enabling researchers to gauge if the model given as an input to the simulation provides a good description of reality. In this thesis, we choose to use the matter power spectrum as the main statistical tool to evaluate the performance of our statistical emulations, so we present next its mathematical definition.

Let us consider a batch of the Universe with volume V_u, and take the average density in that volume to be $\bar{\rho}(t)$. The density at a point specified by the position vector $\vec{r}$ to be $\rho(\vec{r}, t)$, the density contrast/fluctuation is usually defined as:

$$\delta(\vec{x}, t) = \frac{\rho(\vec{x}, t) - \bar{\rho}(t)}{\bar{\rho}(t)} \tag{1.1.7}$$

this quantity measures at each point in the volume V_c, the deviation of the density field from the mean density in the total volume. For sufficiently large (cosmological) volumes, the average background density is simply given by $\bar{\rho}(t) = \Omega_m \rho_c(t)$. It is usually assumed that cosmological fields, like the overdensity field, are ergodic random fields that specified by a set of joint probability distributions of the form $\langle A(x)A(x')\rangle$, $\langle A(x)A(x')A(x'')\rangle$, , ..., $\langle A(x)A(x')...A(x^{(n)})\rangle$ where $\langle A(x)A(x')\rangle$, $\langle A(x)A(x')A(x'')\rangle$, corresponding to the two-point, three-point,... , and n-point correlation functions. The brackets indicate that we are taking the average over an ensemble of Universes. This means that our universe can be regarded as a stochastic realization of an underlying physical mechanism specified by the previous (infinite) set of point correlation functions. The ergodic hypothesis then means that we can replace the ensemble averages with spatial averages over large volumes in our single universe realization. Due to

this assumption the 1-point correlation function of the over-density field will always be zero $\langle\delta(\vec{x})\rangle = 0$.

If we consider an ensemble of Universes where the over-density field δ inhabits, and if we consider that each Universe in the ensemble follows the cosmological principle, then the statistical properties of δ must be homogeneous due to translational and rotational invariance (even if δ itself is a measure of inhomogeneity). This additional assumption of the cosmological principle implies that the over-density 2-point correlation function is solely dependent on distance, $\langle\delta(\vec{x})\delta(\vec{x}+\vec{r})\rangle = \xi(r)$

Now, if we consider a flat, comoving geometry, we can Fourier expand the over-density field δ over a large cube of volume V_u, yielding:

$$\delta(\vec{x},t) = \sum_{\vec{k}} \delta_k(\vec{k},t)e^{-i\vec{k}\cdot\vec{x}} \tag{1.1.8}$$

where,

$$\delta_k(\vec{k},t) = \frac{1}{V_u}\int \delta(\vec{x},t)e^{i\vec{k}\cdot\vec{x}}d^3x \tag{1.1.9}$$

by imposing periodic boundary conditions on the box, the wavenumber k will be restricted to the components $\vec{k} = \frac{2\pi}{L}(n_x, n_y, n_z)$. If we consider our volume V_u to be large enough compared to our regions of interest, then the summation in (1.1.8) can be replaced by an integral multiplied by a pre-factor. Consequently, the two-point correlation function yields:

$$\xi(r,t) = \frac{V_u}{(2\pi)^3}\int \langle|\delta_k(\vec{k},t)|^2\rangle e^{-i\vec{k}\cdot\vec{r}}d^3k \tag{1.1.10}$$

with

$$P(k,t) = \langle|\delta_k(\vec{k},t)|^2\rangle = \frac{1}{V_u}\int \xi(r,t)e^{i\vec{k}\cdot\vec{r}}d^3r \tag{1.1.11}$$

where *P(k,t)* is the power spectrum of the density fluctuations. A typical and useful way to represent the power spectrum is by transforming it into a dimensionless quantity through the expression:

$$P_{adm}(k,t) = \frac{V_u}{2\pi^3}4\pi k^3 P(k,t) \tag{1.1.12}$$

Through this quantity it is possible to calculate the variance of the Gaussian field:

$$\sigma^2(t) = \langle\delta(x,t)^2\rangle = \int_0^\infty P_{adm}(k,t)\frac{dk}{k} \tag{1.1.13}$$

and finally, the primordial power spectrum is described by a power law:

$$P_\delta(k) = A\left(\frac{k}{k_p}\right)^{n_s-1} \tag{1.1.14}$$

where n_s is known as the *scalar perturbations spectral index* and $A = P(k_p)$ is the amplitude of the power spectrum normalized at the k_p scale. Both n_s and A are cosmological parameters, that can be predicted by the inflationary theory.

Now, in analogy to the power spectrum being the Fourier transform of the 2-point correlation function, which expresses the correlation of the field among 2 different arbitrary locations in the configuration space, the Bispectrum is the Fourier transform of the 3-point correlation function, given by $\xi_{3p}(r_1, r_2, \theta) = \langle\delta(\vec{x})\delta(\vec{x}+\vec{r_1})\delta(\vec{x}+\vec{r_2})\rangle$, where $r_1 = |\vec{r_1}|$ and $r_2 = |\vec{r_2}|$. This function expresses the correlation among 3 different locations in the configuration space and it gives further information on the

properties of the density field.

Since the 2-point correlation function gives a complete description of a Gaussian field, the 3-point correlation function should be the lowest order statistical tool to probe non-gaussianities in the over-density field. This reasoning makes the latter quite useful since gravitational instability in the non-linear regime could induce departures from Gaussianity, even if we are assuming Gaussian initial conditions. Consequently, these departures can generate signatures in the matter distribution, which can be probed by the Bispectrum/3-point correlation, enabling us to constrain the models of structure formation and the dynamics and nature of gravity and dark matter.

In this thesis we also use the over-density Bispectrum, B, to compare between N-body density fields and the ML emulated ones. We take the usual cosmological definition of bispectrum, see e.g., [12],

$$\langle \delta_{\vec{k}_1} \delta_{\vec{k}_2} \delta_{\vec{k}_3} \rangle \equiv \delta_D(\vec{k}_1 + \vec{k}_2 + \vec{k}_3)\, B(k_1, k_2, k_3), \tag{1.1.15}$$

where δ_D is the Kronecker delta function. We will use a configuration of wavenumbers often assumed in simulations (and observations), where two scales have fixed magnitudes and the third one varies according to the angle between the scales with fixed magnitudes.

Now, since one of the building blocks of this work consists in N-body simulations, it is important to introduce the key ideas and briefly describe the main techniques used in them. This is what we will do in the following Section.

1.1.3 N-body Simulations

N-body simulations are a powerful numerical tool that can be used to find answers to the main questions of structure formation. One of such questions is to understand how such small perturbations in the primordial Universe result in the complex and diverse pattern of macro-scale structures (e.g. galaxies, clusters, and filaments) we see today.

At the end of the first half of the XX century, Lifshitz applied the Jeans theory of gravitational collapse to an expanding Universe, describing the evolution of inhomogeneities in a Universe with an FLRW background through the use of linear perturbation theory [13]. Nevertheless, as mentioned earlier, this theory only works while perturbations are small. With their growth, non-linear evolution starts to dominate, and more sophisticated methods became necessary to follow their evolution. Numerical N-body simulations do the job by simulating the dynamics of gas (baryonic matter) and dark matter with a consistent set of dynamic equations coupled by gravity for both components. They are particularly reliable due to the minimal amount of assumptions required to perform them, and integration errors that can be limited to any required level of resolution.

The procedure to obtain a set of density field snapshots from an N-body simulation run is usually separated into two main steps: the setting up of the initial conditions snapshot and the time step integration of the equations of motion, that evolved the initial distribution of particles positions and velocities to any time in the future. We will now briefly describe these two steps.

Initial Conditions: To set up N-body simulation initial conditions, two main specifications need to be made, the background cosmology and the initial perturbations to this background. The background cosmology can be specified through the cosmological parameters introduced in (1.1.6). The perturbations are assumed to be small and well described by linear perturbation theory at initial times. In these conditions, they can be computed using the Zel'dovich approximation [14], which allows generating the particles' initial displacement field according to the following equations involving the comoving position

and peculiar velocity of particles at a time t:

$$\vec{x}(t) = \vec{q} + D(t)\vec{u}(\vec{q}); \qquad \dot{\vec{x}}(t) = \dot{D}(t)\vec{u}(\vec{q}) \tag{1.1.16}$$

where $\vec{x}$ and $\vec{q}$ are the final and initial comoving coordinates and the dependency on the perturbations is stored in the initial velocity displacement field, $\vec{u}$, through $\delta(q) = -\nabla \cdot \vec{u}(q)$, which is assumed to be irrotational ($\nabla \times \vec{u} = 0$), and also on the growing mode factor $D(t) \propto g(\Omega, \Omega_\Lambda)t^{\frac{2}{3}}$. Here $g(\Omega, \Omega_\Lambda)$ is known as the linear growth suppression factor, which gives the rate of growth of the density perturbations relative to the growth in an Einstein–de-Sitter Universe.

After having both the cosmological model and the initial perturbations specified, the positions of the mass particles are initially placed on a regular cubic grid with N^3 points inside the simulation's volume $V = L^3$. The perturbation δ is then computed in Fourier space, assuming periodic boundary conditions (with the exclusion of the Nyquist modes), from a random realization of the matter power spectrum, $P_{lin}(k, t_i)$, assuming linear perturbation theory. The initial time, t_i, is usually chosen at a high enough redshift, e.g. $z_i > 50$, to make sure perturbations are still well approximated by linear evolution on all scales resolved in the simulation's box. (i.e. z_i is usually set according to the evolutionary state of the smallest-resolved scale $\sim L/N$ in the simulation).

Finally, the initial displacements and velocities are calculated through the Zel'dovich approximation equations (1.1.16). These positions and velocities will then serve as an input (the initial conditions snapshot) of the N-body integrator code, which we will briefly describe next.

The Particle Motion: Interestingly enough, although these tools probe cosmological scales where general relativity dominates, the gravitational potential is evaluated, in most codes, with Newtonianian theory. This can be justified because large-scale structure does not involve strong gravitational fields, and because of Birkhoff's theorem [15], which states that any spherically symmetric solution of the vacuum field equations must be static and asymptotically flat. This means that the space inside a spherical volume smaller than the curvature radius must be flat and unaffected by the matter distribution outside the sphere. Having these in mind, N-body gravity codes with pressureless components (e.g. collisionless dark matter) usually implement the following Newtonian evolution laws:

$$\frac{d\vec{x}}{dt} = \frac{1}{a}\vec{v} \tag{1.1.17}$$

$$\frac{d\vec{v}}{dt} + H\vec{v} = -\frac{\nabla\phi}{a} \tag{1.1.18}$$

$$\nabla^2\phi = 4\pi G a^2[\rho(\vec{x}, z) - \bar{\rho}(t)] \tag{1.1.19}$$

where $\vec{v}$ is the peculiar velocity and $-\frac{\nabla\phi}{a}$ the peculiar acceleration. To obtain the evolution of the particle trajectories in time, one needs to integrate these equations and compute the potential ϕ for all the individual particle positions, which will consequently depend on the positions and masses of all the remaining particles. Equations (1.1.17) - (1.1.19) are also complemented by equations granting the conservation of energy/entropy, momentum, and mass (the latter is usually granted by construction).

There are three main methods used in traditional N-body codes to compute the gravitational potential. The *Particle-Particle* (PP) algorithms consider all the particles individually, computing the masses and positions of every single one of them. The *Particle-Mesh* (PM) algorithms approximate the force acting on a particle by the average potential generated by a set of neighbourhood particles. Finally, the *Particle-Particle-Particle-Mesh* (P^3M) algorithms consist of a hybrid approach that uses the two previously mentioned algorithms, the PP algorithm for short-range forces and the PM algorithm for long-range

ones. We can expect that although more precise, the PP algorithm should be the most computationally expensive since it requires the calculation of the gravitational potential for every single point particle in the grid. Concerning the last two, the computation power required is lower since we are dealing with approximate calculations of the potential [15–18].

The simulations used in this thesis were generated with the `Hydra` code [19], which implements an adaptive mesh refinement P^3M method to compute gravitational forces. The code adopts a predictor-corrector time integration scheme to evolve particle positions and velocities forward in time. The initial conditions box was also generated with the code `cosmic` from the `Hydra` software package. It implements the Zel'dovich approximation described earlier and the Bardeen-Bond-Kaiser-Szalay (BBKS) cold dark matter transfer function to compute the initial conditions power spectrum. In this way is possible to obtain simulation snapshots at different redshifts. Since `Hydra` is a *lagrangean* code (that follows particles at their positions instead of assuming an *eulerian* regular grid description) and the statistical learning methods that we will use in the thesis requires the evaluation of densities on a regular grid, we mapped particles positions into a regular grid of densities by using a mass assignment scheme that uses smooth particle hydrodynamics and dark matter particle densities computed with the `darkdens` code from the `Hydra` package [19, 20].

1.2 Gravitational Waves

One of the most exciting predictions of the last century in cosmology was Gravitational Waves (GW). This prediction was inferred from Einstein's General Theory of Relativity and was much later confirmed in 2015, by the LIGO detectors [21]. This opened the doors to a new era for Astrophysics and Cosmology where the study of the Universe is no longer exclusively constrained to the use of electromagnetic radiation. We still do not fantom the full scientific potential of this new astrophysical messager, but we do know some incredible applications which make it a very powerful tool to provide new insights on a vast range of astrophysical and cosmological themes.

First of all, since Gravitational Waves (GWs) are a natural consequence of the interaction between matter in a relativistic context, their detection and study give means to test and constrain the theory of General Relativity [22, 23]. Secondly, gravity is the strongest force at cosmological scales, it couples "minimally" to matter and radiation, and propagates freely as soon as it is generated. This fact enables us to probe the early universe at ages far outside the reach of typical electromagnetic detectors, increasing the potential for scientific discoveries in the fields of cosmology and fundamental physics [24]. Lastly, gravitational waves can only be generated by an asymmetric mass source with a time-varying quadrupole moment, which includes astrophysical phenomena such as supernovae explosions, asymmetric rotating compact stars and compact binaries and thus the detection of GWs originating from these sources has the potential to enable the improvement of the current models which describe them [25].

In a nutshell, Gravitational Waves are a revolutionary tool to do astrophysics and cosmology and have the potential to provide paradigm-shifting discoveries in the near future.

Next, we present a short theoretical introduction on gravitational waves. We will begin by introducing the linearized theory of gravity, showing how it gives birth to the GW prediction. Following that, we will introduce the Multipole Expansion in the context of GW generation and justify our use of the Chirp Mass as the free parameter for our regressions. Finally, we close this theme by discussing the stages of gravitational collapse, and the methods used to incorporate them in the waveform calculations. We refer the reader to [26] for more details about the contents of next Section.

1.2.1 The Linearized Theory of Gravity

The prediction of gravitational radiation stems from linearizing Einstein's equations and choosing a proper gauge.

If we are far away from any given mass distribution we can consider the metric to be a small perturbation h_{ab} from the flat Minkowski metric η_{ab}, yielding:

$$g_{ab} = \eta_{ab} + h_{ab} \tag{1.2.1}$$

if we work exclusively in linear order for h the Riemann tensor will yield:

$$R^{a}{}_{bcd} = \partial_c \Gamma^{a}{}_{bd} - \partial_d \Gamma^{a}{}_{bc} = \frac{1}{2}\left(\partial_b \partial_c h^{a}{}_{d} + \partial_d \partial_a h_{bd} h_{cb} - \partial_c \partial^{a} h_{bd} - \partial_d \partial_b h^{a}_{c}\right) \tag{1.2.2}$$

where the dependency on the metric is contained in the so-called *Christofell connections*, given by $\Gamma^{a}{}_{bc} = \frac{1}{2} g^{ad}\left(\partial_b g_{dc} + \partial_c g_{bd} - \partial_d g_{bc}\right)$. The Ricci tensor and scalar are obtained by contracting and taking the trace of the Riemann tensor, both yielding, respectively:

$$R_{bc} = R^{a}{}_{bac} = \frac{1}{2}\left(\partial_c \partial_a h^{a}_{b} + \partial_b \partial^{a} h_{ca} - \partial_b \partial_c h - \Box h_{bc}\right) \tag{1.2.3}$$

$$R = R^{b}{}_{b} = \partial^{a} \partial_d h_{a}{}^{d} - \Box h \tag{1.2.4}$$

where the notation $\Box$ corresponds to the D'Alembertian operator $\Box = \partial_a \partial^a$ and h is simply the trace of the perturbed metric $h = h^{a}{}_{a}$.

Now, plugging in these expressions in the Einstein tensor and defining the trace-reversed perturbed metric as $\bar{h}_{bc} = h_{bc} - \frac{1}{2}\eta_{bc} h$, we can arrive at the following linearized form for the tensor:

$$G_{bc} = \frac{1}{2}\left(\partial_a \partial_c \bar{h}^{a}{}_{b} + \partial^{a} \partial_b \bar{h}_{ca} - \Box \bar{h}_{bc} - \eta_{bc} \partial_a \partial^{d} \bar{h}^{a}{}_{d}\right) \tag{1.2.5}$$

although this expression already corresponds to Einstein's tensor in its linearized form, it is still not obvious how it can be interpreted/translated into a wave theory of gravity.

In order to obtain something similar to a wave there is still the need to choose the so called *Lorenz Gauge* (in analogy to electromagnetism). The Lorenz Gauge corresponds to a system of coordinates satisfying $\partial^{b} \bar{h}_{bc} = 0$, which is referred as the *Lorenz condition*. If we apply this condition to the previously obtained form of Einstein's tensor we obtain:

$$G_{bc} = -\frac{1}{2}\Box \bar{h}_{bc} \Rightarrow \begin{cases} \Box \bar{h}_{bc} = -\frac{16\pi G}{c^4} T_{bc} & \text{(sourced)} \\ \Box \bar{h}_{bc} = 0 & \text{(vacuum)} \end{cases} \tag{1.2.6}$$

for the case of a gravitational wave sourced by a matter/energy distribution modelled by the energy-momentum tensor T_{bc} (above) and for the case of a gravitational wave propagating in vacuum (below).

And the wave nature of the gravitational perturbations becomes clear, if we recognize (recalling the form for the D'Allembertian operator) these expressions as wave equations where $\bar{h}$ corresponds to the amplitude of the wave, which in our work and from now on, will be called the *strain* of the GW.

1.2.2 Generation of GWs and Binary Systems

Since our work with gravitational waves concerns binary inspirals, it is useful to study the solutions for the sourced form of the linearized Einstein equation:

$$\Box \bar{h}_{bc} = -\frac{16\pi G}{c^4} T_{bc} \tag{1.2.7}$$

In the following text, we describe some important elements needed to find solutions for this equation.

One important step is choosing a new gauge, the so-called *Transverse-Traceless gauge* (TT gauge). This gauge incorporates the Lorenz condition ($\partial^b h_{bc} = 0$), and two additional ones, the traceless condition ($h = h^a{}_a = 0$) and finally a condition eliminating all the temporal components ($h_{a0} = 0$). The first condition (transverse) imposes a transverse propagation of the perturbations and eliminates 3 degrees of freedom, the second (traceless) eliminates one additional degree of freedom. Finally, the third condition eliminates 4 degrees of freedom through the imposition of having all temporal components nullified. We know, through the imposition of isotropy and homogeneity (cosmological principle) that our metric has an initial 10 degrees of freedom, so summing up, in the TT gauge we will have only two degrees of freedom remaining, which correspond to the two different polarization states of the GWs. We have them usually called "plus" polarization (h_+), which induces elongations and contractions in space-time along the x and y-axis and the "cross" polarization ($h_\times$), which induces a tilt effect by adding a transverse component. Now, keep in mind that for matter-sourced spacetimes, due to the presence of non-radiative degrees of freedom, it is not possible (in general) to write the metric perturbation in the TT gauge. However, the perturbation can be split into unique pieces, where the radiative degrees of freedom are satisfied by the so-called TT piece.

Returning to the theme of finding a solution for (1.2.7), one does it by applying a Green's function. The field generated by the source is given by integrating the Green's function against the source function. Following that, the evaluation of the spatial part of the metric perturbation at large distances from the source gives the first term in the multipolar expansion of the radiation field. Imposing the conservation of the energy-momentum tensor and defining the second-order momenta of the mass distribution gives a differential equation relating the latter with metric perturbation. Finally, by projecting the non-TT parts of the metric into the TT gauge, one can obtain:

$$h_{ij}^{TT} = \frac{4G}{rc^4} \Lambda_{ij,kl}(\hat{n}) \ddot{M}^{kl}(t - \frac{r}{c}) \tag{1.2.8}$$

where G and c are the gravitational constant and light speed, r the distance to the source, $\Lambda_{ij,kl}$ the operator used for the projection in the TT gauge and finally M^{kl} is the second-order momentum of the energy density (T^{00}). In addition to the aforementioned reference, [26], we refer the reader to [27] for further details on the derivation of Eq. (1.2.8).

This expression is a remarkable result because it shows that the leading order term of the gravitational multipole expansion is a quadrupole. Consequently, to obtain gravitational radiation, one has to have at least a time-varying quadrupole moment, which differs from our knowledge of electromagnetic radiation.

One such system, capable of having a time-varying quadrupole moment is a massive binary system, the more massive it is, the higher is the strain of the gravitational waves it emits. Currently, only very massive compact systems such as Binary Black Holes (BBHs) or Binary Neutron Stars (BNS) are capable of emitting gravitational radiation with strains large enough to be detected by our technology.

From Quadrupole Formula, Eq. (1.2.8), it is possible to obtain a simple approximation of the expressions for the strain of a waveform emitted by a massive binary system, in both polarizations. If we

assume that we have a circular orbiting system, with masses m_1 and m_2 and a given orbital motion, if we additionally ignore the proper motion of the system and any back-reaction on its motion due to GW emission and assume that it is lying in the (x, y) plane. Calculating the second-order mass momenta in the center of mass of the system it is possible to obtain expressions for the strain's "plus" and "cross" polarization modes of the gravitational waveform:

$$h_+(t;\theta,\phi) = \frac{4}{r}\left(\frac{GM_{chirp}}{c^2}\right)^{\frac{5}{3}}\left(\frac{w_s}{c}\right)^{\frac{2}{3}}\frac{1+\cos^2(\theta)}{2}\cos(2w_s t_{ret}+2\phi) \tag{1.2.9}$$

$$h_\times(t;\theta,\phi) = \frac{4}{r}\left(\frac{GM_{chirp}}{c^2}\right)^{\frac{5}{3}}\left(\frac{w_s}{c}\right)^{\frac{2}{3}}\cos(\theta)\sin(2w_s t_{ret}+2\phi) \tag{1.2.10}$$

where (θ,ϕ) are the angular coordinates of the system, w_s is its angular frequency, $t_{ret} = t - \frac{r}{c}$ the retarded time and finaly M_c is the Chirp Mass of the system, given by:

$$M_{chirp} = \frac{(m_1 m_2)^{\frac{3}{5}}}{(m_1+m_2)^{\frac{1}{5}}} \tag{1.2.11}$$

This parameter is extremely useful due to its relation to the frequency evolution of the GW in its inspiral phase:

$$M_{chirp} = \frac{c^3}{G}\left(\left(\frac{5}{96}\right)^3 \pi^{-8} f_{GW}^{-11}(\dot{f}_{GW})^3\right)^{\frac{1}{5}} \tag{1.2.12}$$

where f_{GW} is the frequency of the gravitational wave. Keep in mind that this expression is only valid in the Newtonian approximation and to disentangle the two masses one needs to include higher-order terms of the Post-Newtonian formalism, which will be briefly discussed in the following Subsection.

Finally, since this expression implies that the Chirp Mass can be directly obtained from observational data, we invoke it to justify our specific use of this quantity as the free parameter in our work on GWs.

1.2.3 The Waveform: Inspiral, Merging and Ringdown

Even though we have a solution for the amplitude of gravitational waves emitted by a binary source, this solution does not include all physical effects that occur in reality. The previously described treatment to obtain the quadrupole formula assumes that the space-time curvature and the velocity of the source are independent, meaning that it should only apply to a system governed by non-gravitational forces. Moreover, it also only applies to a non-relativistic context, where the internal velocity of the source is still small.

Now, compact binary systems are considerably outside of those regimes for two reasons. First, they are compact, which means that they have non-negligible self-gravity, and thus the system will be governed by gravitational forces. Second, as the system approaches the so-called *merging* stage the velocities increase to the relativistic regime. Thus, to describe the evolution of the system precisely, there is a need to rely on approximate methods and expansions of Einstein's equations and GW multipole formula. The most widely used of these methods is the so-called Post-Newtonian approximation, which is called Post-Newtonian in the sense that it applies to weakly gravitating and slowly moving sources, although it still seems quite a constrained regime it has been shown that it is enough to describe with reasonable precision most of the evolution of the binary dynamics, during the so-called *inspiral* phase. These methods start with a Newtonian description of gravity and progressively add relativistic corrections, expanding the equations of motion around a small parameter which expresses the deviations from the classical regime.

This parameter is usually defined as $\epsilon_{PN} = \frac{v}{c}$, where v is the internal velocity of the source and c the speed of light [28]. As we can see from the definition, the closer this parameter approaches unity, the closer we are from a purely relativistic regime. Moreover, given the fact that the quadrupole formula is considered to be of leading order in the PN formalism, since it applies to cases where $\frac{v}{c} \to 0$, we must not limit ourselves to it. Once the motion of the source is computed to the desired PN order, the multipole expansion must be expanded to the same desired order.

As the system evolves and keeps losing energy by gravitational radiation, the two bodies eventually plunge into each other and undergo merging progressively stabilizing into equilibrium during the third and final phase called the *ringdown*. To describe these two last phases, even the PN formalism breaks down and other methods, numerical and perturbative, must be applied.

In summary, the three phases of gravitational wave emission, and description of the full waveform may be treated using the following regimes:

1. **Inspiral:** This is a mildly relativistic regime, where the system undergoes adiabatic inspiralling and the signal can be appropriately described by PN approximations.

2. **Merger:** Here the system enters into a strong relativistic regime. It undergoes an unstable plunge where the two objects plunge into each other and collapse to form a Black Hole. Numerical relativity is the appropriate tool to compute the merging of the binary into a black hole. Matching between PN and NR waveforms has been solved using hybrid models such as the *Inspiral-Merging-Ringdown* (IMR), which parametrizes the domain between the PN and NR regimes in a phenomenological way. It is also possible to describe the merging with a resummation technique called the *Effective-One-Body* (EOB) formalism, which recasts the two-body dynamics in a simpler one-body model using the results of PN theory, black-hole perturbation theory, and the gravitational self-force formalism.

3. **Ringdown:** This is the final regime. The result is a newly formed Black Hole in a stationary configuration given by the Kerr solution. During the relaxation, the perturbed Black Hole emits the so-called quasi-mode radiation and these modes are described by Black Hole perturbation theory.

The LAL Simulation library used by the PyCBC platform includes a variety of approximants that give the final waveforms using approximations of the methods described above, sometimes using hybrid models coupling the methods together.

From all the approximants included in the LALsim library we chosed to use the **SEOBNRv4_opt** [29] approximant to work in Time Domain and the **TaylorF2** [30] to work in Frequency Domain. The *SEOBNRv4* approximant uses EOB formalism calibrated to Numerical Relativity simulations, while the *TaylorF2* approximant can be directly computed from the PN time-domain approximant *TaylorT2* using the stationary phase approximation.

1.3 The role of Statistical Learning

In the previous Sections, we summarised the key features of N-body simulations and GW waveform approximators and their importance for Cosmology and Astrophysics. Although these are accurate and mature/sophisticated methods, their use and posterior scientific analysis still bring drawbacks. This work attempts to apply statistical learning methods to mitigate some of these drawbacks.

Regarding structure formation, although N-body simulations can give us complete descriptions of the evolution of the density fluctuations, they have the downside of being computationally demanding,

requiring large amounts of memory and CPU processing times, both quantities scaling with the size of the simulation and the number of mass particles considered.

To try to solve this problem we propose to apply statistical learning methods to reduce the time and computational resources needed to obtain density field realizations with accuracy as similar as possible to that of a true N-body simulation. Given an initial set of N-body simulation outputs, it should be possible to train statistical learning algorithms to learn the distribution of densities in those simulations and fit regression models which relate them with a specific choice of cosmological parameters. Once those regression models are built, the process of obtaining a new density field for a new set of cosmological parameters is just a matter of applying a fast estimation algorithm based on the trained model.

The main problem with this approach is the large amounts of data contained in a typical N-body dark matter simulation density vector, which can make the statistical learning algorithms task extremely time-consuming and memory demanding, and their regression models may also show non-optimal accuracies. On the other hand, there should be a lot of "redundancy" contained in the Universe's large scale structure (in part due to homogeneity and isotropy), and thus it should be considered "compressible". To overcome the computational issues and profit from this compressibility idea, we apply these methods in a way that allows obtaining fast and accurate N-body outputs without wasting as much time as the N-body simulations themselves. We first compress the N-body outputs training data through a simple basis transformation, in this case a PCA, and instead of providing the full output of the simulations as training data for our algorithms, we provide the compressed data representation.

Here we will demonstrate that our statistical learning methodology can provide a new way of obtaining cosmological density fields in a much faster and efficient way, which can then be compared (via statistics such as the power spectrum, correlation functions, and mean background properties) with the observed density field in our Universe.

There have already been examples of successful application of the concept of "compression" of cosmological information. The Euclidemulator [31], has used *Principal Component Analysis* to build an emulator of the non-linear dark matter power spectrum to achieve unprecedented emulation precision. The emulator was constructed from training the model on the PCA components of the power spectra of a set of N-body simulation runs. A somewhat similar approach to ours is the case of [32] that applied statistical learning methods to show that is possible to make parameter inferences given a dark matter density field. Another approach closely related to ours is the case of [33], which used statistical learning methods to emulate dark matter density fields, with the difference that they use the "initial conditions" displacement field as an input parameter to the machine learning methods, instead of working directly with the cosmological parameters.

Regarding gravitational waves, the approximate methods used for the generation of waveforms based on a set of astrophysical parameters concerning the GW generating system are very fast and accurate. It should also be possible to apply the reasoning above and attempt to produce even faster results with reasonable accuracy through the use of statistical learning. However, in this work, we aim to do the opposite, instead of emulating a GW waveform given a set of astrophysical parameters we will attempt to perform accurate predictions of the parameters given the GW waveform.

Most techniques for parameter inference in the GW context use Bayesian inference and thus are likelihood dependent, it is interesting to try to alleviate that dependency by switching to a paradigm of inference through statistical learning methods, which do not rely on a priori assumptions – except for the creation of the training set – and are less subject to systematic errors, as is the case of likelihood parameter searches.

Several works have been performed in the application of these methods to the inference of the param-

eters associated with binary GW emitting systems. As examples, we have the work in [34], where they apply Deep Neural Networks for both GW detection and parameter inference based on real LIGO data. In [35] and [36], they also use Deep Neural Networks for parameter inference but work on a reduced-order basis instead of the full waveform strain vector, which is in a way similar to our compression step using PCA.

One of the main innovations of our work is the combination of simple supervised models and PCA compression to achieve similar or better results than more complex and memory demanding methods based on Deep Neural Networks. We chose to apply four supervised learning methods, that are more amenable for interpretation, to compare their performances and gauge their pros and cons. The algorithms we chose to use in our analysis are the Random Forest, Extremely Randomized Trees, Support Vector Machines, and single-layer Neural Networks (NN).

Additionally, we implement the training step using waveforms with no noise added and test the algorithms in data with signal *plus* noise, as opposed to some of the mentioned works, which train the algorithms already with signal *plus* noise waveform representations.

In this way, our algorithms might be further adapted to attain a higher generalization capacity, working together with different noise PSDs without requiring the implementation of complex and instrument dedicated networks such as the aforementioned Deep NN's methods.

Chapter 2

Statistical Learning Methods

The two most widely known statistical learning branches are called supervised and unsupervised statistical learning. Both approaches are extremely powerful due to their capabilities to model and retrieve information from large datasets. Given a data set containing variables with no explicitly defined model, these approaches are capable, for example, of learning the relationship between variables and perform approximate regressions or compressions into lower-dimensional spaces.

Both cases deal with data sets of n observations, each observation spread across a certain number of m dimensions, usually called *features* in the data science context.

In the case of supervised machine learning, we consider not only the observations and their dimensions, but also a target variable, attributed to each observation, and the machine learning algorithms will receive as an input not only our n observations across the m dimensions but also the n target values attributed to each of them. The algorithm will then find a map, by partitioning the m-dimensional space, which relates the distribution of the feature and target variables. This map is usually called the machine learning *model*, and it can thus predict a new target variable for an observation at not previously observed parts of the same m-dimensional space, which wasn't previously included in the training process.

As for unsupervised machine learning, the main difference to the just described branch is that there is no target value involved, only the observations and corresponding dimensions are given to the algorithm. Following that and usually some distance metric, the algorithm will find patterns, groups/clusters or other kinds of new representations of the data, without fitting any regression model or requiring a posterior prediction. Principal Component Analysis (PCA) is considered a method of unsupervised machine learning since it does not involve any kind of prediction or fitting to a target value. This new representation can be truncated, since each principal component is ordered by the total variance of the dataset, thus possibly resulting in a lower-dimensional representation of the data if the data is linearly compressible.

One additional important aspect of this work is the so-called "optimization" of the algorithms. Each statistical learning method depends on a set of specific parameters which are called the *hyper-parameters* of the model. Depending on their values and on the nature of the dataset, the performance of the model may vary considerably. The optimal hyper-parameters depend on the dataset we have, the number of features and their nature, and it is not straight forward to deduce which values should be chosen for each case. The usual approach is to choose the hyper-parameters in a "trial and error" process, where a set of models are built considering a range of possible hyper-parameters and the ideal choice is the one that gives the best performance error, given an appropriate error measure.

A great deal of work and time is spent in this process since different optimization schemes can be applied, with varying performances and computational requirements. Additionally, since we are using four distinct machine learning algorithms, they need to be individually optimized.

In the following Sections, we will discuss in more detail the methods that we adopted in this work. We begin by describing the supervised machine learning algorithms used in this work, giving some details on their mathematical formulation. Afterwards, we describe the PCA method as well as a continuous version of it called FPCA. Finally, we conclude with remarks on the optimization processes, explaining

the different optimization schemes that were applied.

2.1 Random Forest

One of the methods we adopted in this work is called Random Forests. However, to understand Random Forests [37] it is first required to understand Decision Trees, and thus we will introduce them first.

A Decision Tree [38] consists of a classification or regression model building algorithm, which learns decision rules based on the relationship between the feature values and the target values in the provided training data. They are built by recursive partitioning of the feature space. Each feature domain can be split through a decision process representing a node in the tree, and each node is split recursively until some terminating criterion is fulfilled. Once that criterion is fulfilled, the node becomes associated with a final prediction (binary or continuous depending on classification or regression) and becomes what is called a *leaf node*.

When the model converges, what results is a tree structure where each node represents a logical condition connecting the possible values for the target we want to predict and the corresponding branch, which is an ensemble of parent and child nodes, representing the segmentation of the feature space.

In each node of the tree, a crucial decision needs to made, which is whether the node keeps being split into child nodes, or is assigned with the target value, becoming a leaf and terminating that branch. The criterion used to split a node depends on whether we are dealing with a classification or regression problem. In the case of classification, it is usually the *Information Gain*, given by:

$$\text{Gain} = \text{Info}(D) - \text{Info}_A(D) \tag{2.1.1}$$

where D is the data partition, $\text{Info}(D)$ and $\text{Info}_A(D)$ are the expected information needed to classify an instance in D and to classify an instance in D based on the partitioning relative to the feature A. Which are given by:

$$\text{Info}(D) = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{2.1.2a}$$

$$\text{Info}_A(D) = \sum_{j=1}^{l} \frac{|D_j|}{|D|} \text{Info}(D_j) \tag{2.1.2b}$$

So given the description above, n should correspond to the possible number of values that can be attributed to the target and l to the number of values in which the feature A can be partitioned. Here $|D|$ and $|D_j|$ denote the number of instances in the partition D and the child node/sub-partition D_j, respectively, so they correspond to the weighting term of the sub-partition j. And finally, p_i corresponds to the probability of an arbitrary instance in the partition belonging to the class C_i, given by $\frac{|C_{i,d}|}{|D|}$ where $C_{i,d}$ denotes the number of instances of the class C_i residing in the partition D. So we can conclude that for the case of classification, the splitting criteria always follows the direction of lesser entropy or higher purity.

In the case of regression, since we are dealing with continuous variables, we must choose the splitting criteria accordingly. Usually, it is defined as the *residual sum of squares* (RSS), considering a division of the feature space into the partitions $D_1, ..., D_s$, we have:

$$\text{RSS} = \sum_{s=1}^{S} \sum_{i \in D_s} (y^{(i)} - \bar{y}_{D_s})^2 \tag{2.1.3}$$

where $\bar{y}_{D_s}$ is the mean target value in the for the observations in the partition D_s and y_i is the true *ith* target value in that partition.

In this work, since both the target variable (PCs) and the independent variables are continuous variables, we are dealing with a regression problem, so the splitting criteria must go along the lines of the latter *RSS*.

Now, one disadvantage of decision trees is that they tend to over-fit the data, that is, they tend to create models which have very high performances when predicting values of instances belonging to the training set, but have a low generalization capacity. However, Random Forests overcomes that problem by relying on an ensemble of decision trees, where the final prediction will consist on the majority prediction (in the case of classification) or the mean prediction (in the case of regression) of the ensemble.

The reason why the Random Forests algorithm is much better at preventing over-fitting and generally more robust in the error rates is that in each tree only a limited amount of the features/dimensions are considered, and the choice is random. This way it becomes possible to mitigate the effect of having correlations between features in the data and also eliminate unimportant features which could be considered redundant. There is also an additional trick which strongly prevents over-fitting, which is the fact that in the process of building each tree not only a limited set of features is randomly chosen to determine each split but also a limited set of instances in the data. Each tree considers only a sample (with replacement) of the whole data set, typically ~2/3 of the total number of instances. Thus, the algorithm can also mitigate the effect of outliers and very strongly weighted instances.

As can be inffered from this brief explanation, this algorithm has a strong stochastic nature, even after fixing the ideal hyper-parameters, each implementation will give slightly different results unless we set a specific seed.

For the process of optimization of our Random Forest method, we decided to choose three relevant hyper-parameters: the number of trees/estimators in the ensemble, the minimum size of the terminal nodes and the number of features randomly considered in each split.

Regarding the number of trees, it is expected that a higher number of trees gives better performances, but there is always the downside of the computational cost. As we increase the number of trees in the model, the time spent on computation and the amount of memory needed highly increases while the rate of improvement of the model decreases, eventually reaching a point where the small benefits of increasing the number of trees do not outweigh the computational resources needed to perform the regression.

Regarding the maximum size of the terminal nodes, the smaller the number, the larger will be each tree. This is because we are forcing a continuous splitting until the desired maximum number of observations falls into each terminal node. In opposition to the number of trees, a higher value for this hyper-parameter will result in a lower computational cost. Another important property that we can infer is that higher values will also result in models with more generalization capacity and less prone to over-fitting.

Finally, concerning the number of features considered in each split, a lower value will also result in a better generalization capacity, and mitigate possibile redundancies in the feature space.

2.2 Extremely Randomized Trees

Just like the previously described *Random Forest*, the *Extremely Randomized Trees* algorithm [39] is also a Tree Model that builds an ensemble of Decision Trees, taking their output's arithmetic average as the final output of the ensemble, in the case of regression.

One of the main differences in this algorithm with respect to Random Forests is the fact that in each tree the whole training data set is considered, to minimize bias. Another difference is that each split in the tree is determined randomly, without the need to compute any information gain measure or *MSE*. After all the splits are performed, the algorithm selects the ones which provide the least error in the final regression predictions. Consequently, even if the splits themselves are random, the overall final selection process is not. In this way, in principle the overall variance of the ensemble is minimized, making the algorithm less prone to overfitting than the usual *Random Forest*.

Regarding the subject of optimization, the relevant hyper-parameters we chose were the number of trees in the ensemble, just like in the case of Random Forest, and also the number of random cuts. Concerning the latter, setting this parameter to one implies a fully randomized tree, with its structure uncorrelated to the target variable and the trees minimally correlated with each other. If this value is higher than one, the split that results in the highest performance score is the one chosen to split the node and thus the correlation of each tree to the target variable and consequently the bias, also increases.

One final important remark regards the time taken to build the regressions. Since this algorithm performs the feature space splits randomly, without the need to calculate any performance measure, we should expect it to be considerably faster than Random Forest at building its regression models.

Given these reasons, we add this algorithm to our analysis and evaluate if it is, in fact, a worthy substitute to it Random Forest.

2.3 Support Vector Machines

A *Support Vector Machine* (SVM) [40] is an algorithm also appropriate for both classification and regression, which builds its model by segregating the data in the feature space.

This algorithm maps the instances in an N-dimensional feature space where N is the number of dimensions/features in the data, and segregates the data points by building hyper-planes of $(N-1)$ dimensions.

In the case of classification, the final output of this algorithm will result in the hyper-plane maximizing the distance between data points of different classes. This can be reduced to the problem of maximizing the *margin* of the algorithm, defined as the distance between the hyperplane and its closest data points. These data points are the so-called *Support Vectors*.

Different choices of support vectors lead to different orientations of the hyper-plane. In the case of regression, although we still want to maximize the margin, we also want the data points to lay inside of it. To give more details on regression with *SVMs* let us start with the main objective of the algorithm, which is, for the linearly separable case, to approximate our continuous data $(y_1, x_1), ..., (y_n, x_n)$ with a linear function:

$$f(\vec{x}, \vec{w}) = \vec{w} \cdot \vec{x} + b \tag{2.3.1}$$

where x is the instances in the training data, y the labels associated with each instance, w is the weight vector and b the bias term. In the case of classification this would consist on finding the optimal hyper-plane $\vec{w} \cdot \vec{x} + b = 0$ for segregation of the data points, which would then sum up to the dual problem of minimizing $||w||/2$ (which corresponds to maximizing the margin $m = 2/||w||$) subject to the constrain $y_i(w \cdot x_i + b) - 1 \geq 0$.

In the case of regression this sums up to the primal optimization problem:

$$\min_{w,b} \frac{||w||^2}{2} + C \sum_i (\xi_i^+ + \xi_i^-) \quad \text{subject to} \quad \begin{cases} y_i - wx_i - b \leq \epsilon + \xi_i^+ \\ y_i - wx_i - b \geq -(\epsilon + \xi_i^-) \\ \xi_i^{\pm} \geq 0 \end{cases} \tag{2.3.2}$$

in this case, ϵ serves as a margin of tolerance. Since there is a need to account for errors, the slack variables $\xi_i^{\pm}$ for the upper and lower constraints were introduced. The parameter C is the regularization constant, which is a positive number controlling the penalty imposed on observations that lie outside the margin ϵ.

The estimations performed by the algorithm are evaluated through a loss function of the form:

$$L_\epsilon(y, f(\vec{x})) \begin{cases} 0 \quad , & |f(\vec{x}) - y| < \epsilon \\ |f(\vec{x}) - y| - \epsilon \quad , & \text{otherwise} \end{cases} \tag{2.3.3}$$

so, any error within distance ϵ is assumed to be zero. Otherwise, the errors correspond to the distance between the observed values y and the boundary ϵ. The easiest way to find solutions to the primal optimization problem is through the use of Lagrangian Multipliers, which is called the Lagrangian dual formulation. Introducing the Lagrange multipliers α_i^+, α_i^-, the dual optimization problem assumes the form:

$$\max_{\alpha^{\pm}} -\frac{1}{2}\sum_{i,j}(\alpha_i^+ + \alpha_i^-)(\alpha_j^+ + \alpha_j^-)x_i x_j - \epsilon\sum_i(\alpha_i^+ + \alpha_i^-) + \sum_i y_i(\alpha_i^+ + \alpha_i^-) \quad \text{subject to} \quad \begin{cases} \sum_i \alpha_i^+ + \alpha_i^- = 0 \\ 0 \le \alpha_i^{\pm} \le C \end{cases} \tag{2.3.4}$$

solving this problem determines the values for the Lagrangian multipliers and consequently indicates which are the support vectors and their associated errors. The new values are estimated using the function:

$$f(x) = \sum_{i=1}^{N}(\alpha_i^+ + \alpha_i^-)(x_i \cdot x) + b \tag{2.3.5}$$

In the case of non-linearly separable data, it is required to define a kernel, which is a non-linear function that acts as a substitute for the dot product $(x_i \cdot x)$. This process is usually called the *Kernel Trick* [41], mapping the non-linearly separable data into a higher dimensional space where it is possible to find the proper hyper-plane for data segregation. So, in this case, we will have:

$$f(x) = \sum_{i=1}^{N}(\alpha_i^+ + \alpha_i^-) \cdot K(x_i, x) + b \tag{2.3.6}$$

where K is the so-called *Kernel* defining the dot product in the transformed space.

Regarding this work, the support vector machine algorithm implemented [42] enables the use of a linear kernel ($K(x_i, x_j) = x_i \cdot x_j$), a sigmoid kernel ($K(x_i, x_j) = tanh(\gamma x_i \cdot x_j + r)$) , a polynomial Kernel ($K(x_i, x_j) = (\gamma x_i \cdot x_j + r)^{degree}$) and a radial basis kernel ($K(x_i, x_j) = e^{-\gamma|x_i - x_j|^2}$), where the factors γ and r are constraints subject to optimization and thus are treated as hyper-parameters of the model. We will show and justify which kernel we selected as the best for our purposes in Chapter 4, where we present the optimization results.

2.4 Neural Networks

A Neural Network is the oldest and most common of the machine learning methods used in this work [43]. This algorithm is appropriate for regression tasks and can be seen as a "universal function approximator". Just as any curve can be described by an infinite set of straight-lines a neural network can approximate any function by an (ideally) infinite set of basic constituents called the *perceptrons*, ordered in structures called the *Hidden Layers*.

If we consider that we have a dataset with n independent variables $(x_1...x_n)$ associated with m dependent variables $(y_1...y_m)$, a perceptor consists in a mathematical object which receives the value of each one of the independent variables, assigns a weight $(w_1...w_n)$ to them and given a bias variable $(b_1...b_i)$ and an activation function f, computes the output given by:

$$Y_j = f(\sum_{i=1}^{n} w_i \cdot x_i + b_i) = f(z) \tag{2.4.1}$$

so, in a way, we can look at a neural network as a congregation of linear regressors with an output value transformed through a certain activation function. Typical activation functions are the identity/linear $(f(z) = z)$, hyperbolic tangent $(f(z) = \tanh z)$, sigmoid $(f(z) = (1 + e^{-z})^{-1})$ and relu function $(f(z) = \max(0, z))$ and depending of the kind of problem that we are dealing with, the kind of variable that we want to predict or the depth of the network, some activation functions may be more appropriate than others.

The architecture of a whole neural network comprises mainly 3 elements. *The input layer* corresponding to all the independent variables x_i, *the hidden layers*, corresponding to a fully connected set of groups of perceptrons where each group corresponds to one layer and each one of the perceptrons in one layer is connected to each one of the perceptrons in the following layer in a way that the output of each perceptron in a layer k serves as an input to each perceptron in the layer $k + 1$, just as in the way explained in the previous paragraph but now x_i corresponds to Y_j of a given perceptron in layer $k - 1$. Finally, the third element consists of the *output layer*, comprised of just one perceptron in the case of single class prediction or many in the case of multi-class prediction. The number of layers and the number of perceptrons in each layer are set by the user, set to be the values giving minimal error (e.g. MSE for regression, accuracy for classification), against a chosen test set.

To train the neural network to learn the best possible set of weights for a given dataset, two among the most used and classical methods are the *Gradient Descent* [44] and *Backpropagation* [45]. To apply these methods, it is required to first define a *Cost Function*, which will be the error measure on the training set. In the case of regression the MSE yields:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{2.4.2}$$

where θ corresponds to the weight, m the size of the dataset, $h(\theta(x_i))$ corresponds to the ith target value predicted by the network and y_i corresponds to the actual target value.

Gradient descent minimizes the cost function by starting at a random point in the weight space (in practice a random vector of weights is defined at the first iteration), and moving the weights in the negative direction of the gradient of the cost function with respect to themselves until an ideally global (but usually local) minima is reached. The weight update algorithm for gradient descent yields:

$$\theta_j \leftarrow \theta_j - \alpha \partial_{\theta_j} J(\theta) \tag{2.4.3}$$

where α defines the step size at each iteration and is called the *learning rate*. In the case of using MSE as the cost function, it is easy to show that the algorithm translates to:

$$\theta_j \leftarrow \theta_j \frac{\alpha}{m} \sum_{i=1}^{m} [(h_\theta(x_i) - y_i)x_i] \tag{2.4.4}$$

and therefore, at each iteration, the algorithm forces the choice of the weights which further minimizes the cost function.

It is relatively simple to find the expression for the gradient of the MSE with respect to the weights, but when we have a multi-layer network with distinct activation functions, the calculation of their gradient is not trivial. Moreover, there is a need to find ways to propagate the error backwards until we reach the initial layers. That is done to avoid redundant weight assignments and to make every layer "aware" of the changes in the following layers. The *backpropagation algorithm* consists in a process of using the chain rule to compute each layer gradient backwards, starting with the loss function gradient in the last layer and ending in the first hidden layer weight assignment which is consequently followed by a new forward propagation iteration. The detailed mathematics behind this method is outside the scope of this thesis but can be reviewed in [46].

2.5 Principal Component Analysis

Since we will be dealing with large data arrays, the task of building the regression models directly on the full data representation becomes computationally expensive, thus hampering the fulfillment of our objective of performing the estimations with minimal computational resources. Also, as mentioned before, we expect that in any data that is not random noise, there should be some redundancy.

Considering that, it would be reasonable to perform some kind of transformation on the data, profiting from the redundancy and thus alleviating the computational required. For that purpose, we opted to utilize a traditional method called *Principle Component Analysis* [47], that although quite old, only in the last few decades it became computationally feasible to large data sets due to larger memory availability.

Principal Component Analysis is not only used for dimensionality reduction but also visualization and interpretation of high dimensional data sets. This method projects multidimensional data into orthogonal axes of maximal variance, called the *Principal Axes*, ensuring that a maximal amount of variance is represented in a minimal amount of dimensions. In a more detailed sense, if we consider that we have a matrix $\boldsymbol{M}_{nxm}$ composed of n observations in an m dimensional space, it is possible to perform Principal Component analysis by an eigendecomposition of the covariance matrix $\boldsymbol{C}_M = \boldsymbol{M}^T\boldsymbol{M}$.

At its essence, PCA is a linear transformation of the original data matrix $\boldsymbol{M}$ into a new matrix $\boldsymbol{N}$, where its dimensions are completely uncorrelated. It is possible to show that this sums up to the problem of diagonalizing the covariance matrix.

If we consider the linear transformation $\boldsymbol{N} = \boldsymbol{L}\boldsymbol{M}$ and imagine that we already have the new representation $\boldsymbol{N}_{nxm}$, we can compute its covariance matrix $\boldsymbol{C}_N = \frac{1}{n}\boldsymbol{N}\boldsymbol{N}^T$ and represent it with respect to the covariance matrix of the original data, using the following manipulation:

$$\boldsymbol{C}_N = \frac{1}{n}\boldsymbol{N}\boldsymbol{N}^T = \frac{1}{n}(\boldsymbol{L}\boldsymbol{M})(\boldsymbol{L}\boldsymbol{M})^T = \frac{1}{n}L(\boldsymbol{M}\boldsymbol{M}^T)\boldsymbol{L}^T = \boldsymbol{L}\boldsymbol{C}_M\boldsymbol{L}^T \tag{2.5.1}$$

where we used the property $(\boldsymbol{A}\boldsymbol{B})^T = \boldsymbol{B}^T\boldsymbol{A}^T$.

Now, the convenient properties of the covariance matrix (e.g. symmetric, squared) make it possible to apply the finite-dimensional spectral theorem. This theorem states that any symmetric matrix with real values can be diagonalized by an orthogonal matrix. Consequently, we have that $\boldsymbol{C}_M = \boldsymbol{O}\boldsymbol{D}\boldsymbol{O}^T$, where $\boldsymbol{O}$ is the orthogonal matrix with the eigenvectors of $\boldsymbol{C}_M$ as columns and $\boldsymbol{D}$ the diagonal matrix with the eigenvalues in the diagonal. So we have that:

$$\boldsymbol{C}_N = \boldsymbol{L}\boldsymbol{C}_M\boldsymbol{L}^T = \boldsymbol{L}(\boldsymbol{O}\boldsymbol{D}\boldsymbol{O}^T)\boldsymbol{L}^T \tag{2.5.2}$$

So, if we choose $\boldsymbol{L}$ to be a matrix with the eigenvectors of $\boldsymbol{C}_M$ as rows, we will have:

$$\boldsymbol{C}_N = \boldsymbol{L}(\boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L})\boldsymbol{L}^T = (\boldsymbol{L}\boldsymbol{L}^T)\boldsymbol{D}(\boldsymbol{L}\boldsymbol{L}^T) = \boldsymbol{D} \tag{2.5.3}$$

and thus that choice automatically diagonalizes the covariance matrix, giving a truly uncorrelated representation of the data.

Summing up, the Principal Components are simply the eigenvectors of the covariance matrix. These principal components consist of linear combinations of the original matrix dimensions, and thus their interpretation is usually not straightforward, eventhough they are usually more amenable to interpretation than non-linear transforms. Now, to perform the dimensionality reduction, it is required to order the principal components according to their eigenvalue magnitudes, which should yield a representation of the data in order of decreasing variance. Following that, the reduction is performed simply by deleting the principal components with lower variance in an ad hoc way.

The way we implement Principal Component Analysis in this work is through an R function, `prcomp`, that instead of performing the typical eigendecomposition applies *Singular Value Decomposition* (SVD) to obtain the principal components. Singular value decomposition has the advantage that it works for any kind of matrix, including non-symmetric ones, as opposed to eigendecomposition which only works for symmetric matrices. The SVD theorem states that any matrix $M_{m\times n}$ can be decomposed into:

$$\boldsymbol{M}_{m\times n} = \boldsymbol{U}_{m\times n}\boldsymbol{\Sigma}_{m\times n}\boldsymbol{V}^T_{n\times n} \tag{2.5.4}$$

where the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ correspond to the eigenvectors of $\boldsymbol{M}\boldsymbol{M}^T$ and $\boldsymbol{M}^T\boldsymbol{M}$ respectively, and $\boldsymbol{\Sigma}$ is a diagonal matrix containing the square root of the eigenvalues, the so-called *Singular Values*. The key here is recognizing that both matrices $\boldsymbol{M}^T\boldsymbol{M}$ and $\boldsymbol{M}\boldsymbol{M}^T$ are quite special since they are symmetric, square, have the same rank r and the same positive eigenvalues. Since they are symmetric, one can choose their eigenvectors to be orthonormal and thus $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ and $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$ where $\boldsymbol{I}$ is the identity matrix. Finally, the solution is standardized by ordering the eigenvectors in order of decreasing variance, or eigenvalue. To connect the dots about PCA we will have the principal components given by the columns of $\boldsymbol{U}\boldsymbol{\Sigma}$ and the principal axis/directions given by the columns of $\boldsymbol{V}$. Finally, an additional important quantity are the so-called PCA *loadings*, which correspond to the principal directions imbued with variance and contain the correlations/covariance between the original variables and the unit scale components. They are given by $\frac{1}{\sqrt{n-1}}\boldsymbol{V}\boldsymbol{\Sigma}$ and can be used to de-project the transformed data into the original representation. Keep in mind that this interpertation only works if we have row observations and column variables, in our original matrix.

So, in a way, singular value decomposition consists of an improved method for finding Principal Components in any matrix, without the imposition of symmetry.

In the case of this work, we will not use Principal Component Analysis in a typical unsupervised way, instead, we combine it with the supervised machine learning algorithms in a method called *Principle Component Regression* [48]. In other words, we will use PCA to perform an unsupervised dimensionality reduction on our data, and then use the Principal Components as the target variables in the case of density field emulation, and the features in the case of Gravitational Waveform parameter inference. We give the details of our implementation in the following Chapter.

2.6 Functional Principal Component Analysis

We saw that it is possible to find a Principal Component vector for each dimension of a given matrix, that vector will be finite with dimensions equal to the number of rows in the matrix. Now, since in many cases the data one deals with is continuous, it is expected that each Principal Component should also be continuous. Consequently, instead of dealing with eigenvectors, one should be dealing with eigenfunctions. Of course, in practice measurements are discrete so PCA will compute the Principal Components as finite and discrete with each Principal Component vector having one score associated with each dependent variable in the data set. If we could somehow obtain a continuous form for the Principal Components from our initial finite data set we would have a Principal Component score for every possible value assumed by our continuous data distribution and thus we could obtain the full distribution of our data just from that initial finite representation. It is indeed possible to find such continuity in the Principal Components through a method called *Functional Principal Component Analysis* (FPCA) [49–51]. Similarly to PCA, where the Principal Components are computed from the eigenvectors of the covariance matrix, FPCA finds the Functional Principal Components by computing the eigenfunctions of the autocovariance operator. In other words, if PCA sums up to the problem of finding **V** such that:

$$\boldsymbol{CV} = \lambda \boldsymbol{V} \tag{2.6.1}$$

where **V** is the eigenvector and λ the eigenvalue of the covariance matrix $\boldsymbol{C} = \boldsymbol{M^T M}$, FPCA reduces to the problem of finding **V** such that:

$$\int C_f(s,t) V_f(t) dt = \lambda V_f(s) \tag{2.6.2}$$

where C_f is the *covariance function* of our dataset, V_f its eigenfunction, s and t the continuous variables.

In a nutshell, there are two main reasons which make FPCA a method with the potential to surpass the capabilities of the typical machine learning methods. First, it preserves the dimensionality reduction potentialities of PCA, since it represents the original dataset in an eigenbasis of Functional Principal Components. Second, it contains the predictive power of the typical supervised methods presented before since it gives a continuous distribution of scores for each Principal Component given an initial finite and hopefully representative dataset.

2.7 Optimization and Evaluation

As mentioned at the beginning of this Chapter, we will not only use multiple and distinct machine learning techniques to compare their regressions, but we will also boost their performance by optimizing them. Moreover, we have chosen to perform the hyper-parameter optimization using three different approaches, *Bootstrap*, *Grid-Search* and *Cross-Validation*, sometimes even using combinations of both. Here we give a brief explanation of those methods.

2.7.1 Grid-search

Grid-search consists of the simplest and most widely used method for hyper-parameter optimization. The principle consists in creating an initial list/dictionary of possible hyper-parameters that we want to evaluate the model with, and define a range of values where we want to test each hyper-parameter. For example, we can choose to optimize the number of trees in a random forest by defining a range of values between

500 and 2000 in steps of 500, so our vector of possible choices will be $h_v = [500, 1000, 1500, 2000]$. To test these values, we split our data set into a training and validation set and create a loop. In each iteration, we fit the model on the training set with one of the values chosen in our previously defined list and test its performance on the validation set through some preferred metric (e.g. MSE for regression). Finally, as you can already guess, the optimal hyper-parameter will be the one which gives the smallest error, when testing the model against the validation set.

2.7.2 Bootstrap

Bootstrap aggregating [52] is a stochastic method that can be used not only for optimization but also to obtain error measurements in models with statistical fluctuations. The principle is similar to that of grid-search in the sense that we use the same dataset to build multiple models with different hyper-parameters and evaluate them against each other. The main differences are in the actual process of splitting the dataset.

In the usual Bootstrap method, there is a need to define two quantities, the *sample size* and the *number of repetitions*. For each repetition, a sample with the defined size is drawn from the initial dataset, with replacement. The regression model is fitted to that sample, with one of the possible choices of hyper-parameters, and we compute our chosen error metric against the so-called *out of the bag* sample. Once we reach the last repetition, we compute the average error and another set of hyper-parameters is defined. The process keeps running until all our desired range of hyper-parameters is covered. At the end of the last iteration, the best hyper-parameters are the ones which give the lowest mean error of all the repetitions.

So, in a nutshell, the algorithm works in the following way:

1. For each hyper-parameter configuration;
   (a) For each repetition;
      i. Draw a sample of *size=sample size* with replacement;
      ii. Fit the model with the chosen hyper-parameters;
      iii. Evaluate the model on the out of the bag sample;
      iv. Save the score;
   (b) Save the mean of the scores
2. Choose the optimal hyper-parameter configuration as the one which yields the smallest mean of the scores.

The bootstrap method chosen in this work takes the sample size as the size of the initial dataset itself, but since we are drawing with replacement the probability of having a sample equal to the original dataset is incredibly low. The out of the bag sample, in this case, will be the original sample.

The act of drawing samples with replacement and repeatedly taking the error for the same set of hyper-parameters while covering different segments of the dataset provides the advantages to this method. If the number of repetitions is sufficiently high, it ensures that we are covering all of the statistical properties of the dataset, without having a strong bias towards any possibly outweighed part of the data. The data can also have inner correlations between features, or even features which are uncorrelated with the target label, which in principle will get less weight in the chosen error, by using this approach. By choosing samples with the same size of the initial data set it ensures that parts of the data which could be contributing heavily to the overall variance in the original dataset are not disregarded.

2.7.3 K-fold Cross-Validation

K-fold Cross-Validation is yet another scheme commonly used in hyper-parameter optimization. However, the user has, in this case, more freedom to determine the trade-off between bias and variance. Here, there is only the need to define one variable, usually called "K" which determines the number of groups in which our initial dataset will be split into. After the dataset divided into "K" parts, one of them is held out for validation and the model is fitted to the remaining "$K - 1$" parts. The process is repeated "K" times and once finished, every group served as a validation set.

So, again, to summarize the process works like this:

1. For each hyper-parameter configuration;
   (a) The data set is randomly shuffled and split into K groups;
   (b) For each group;
      i. Fit the regression model using the remaining groups as a training set;
      ii. Test the model on the current group using a desired statistic;
      iii. Save the score;
   (c) Take the mean of the K scores;
2. Choose the best hyper-parameter configuration as the one which yields the smallest mean of the scores.

This method has some advantages when compared to bootstrap. In bootstrap, it is possible to define a sample size, and given a high enough number of repetitions, it should be comparable to cross-validation. The downside is that it will always be blind to some part of the initial dataset. On the other hand, in cross-validation, since every group is used for both training and validation, every data point will always have some weight considered in the outcome of the method. If we increase the number of folds, we decrease the bias of the model, on the other side we increase the variance and the model loses its generalization capacity leading to the so-called over-fitting.

The usual choice of K is 10, corresponding to the so-called 10-fold Cross-Validation, which gives a relatively unbiased dataset which still preserves a great deal of its variance. In this work, we chose to perform cross-validation with different choices of folds and compare each result. We additionally implement the so-called *leave-one-out cross-validation* (LOOCV), where the number of folds corresponds to the actual size of the original dataset, and thus one observation is left for testing.

Chapter 3

Methodology

This Chapter presents the method we developed for this thesis and the methods we used to generate our training and test data sets. The whole pipeline includes the use of codes in different programming languages and libraries. These include `Fortran`, for generation the N-body simulations and power spectrum computation from 3-dimensional density volumes; `Python`, to generate GW waveforms, noise models, and to compute density field bispectra; and, finally, the `R` programming language [53] for the new statistical learning pipeline, we propose. This includes using R libraries for performing dimensionality reduction, construction of regression models, and parameter estimation software.

Regarding data generation, we used the public version of the `Hydra code` [19] to construct our N-body density simulations. The density power spectrum was computed using the `powmes` code [54] and the bispectrum using the `Pylians3`[1] `Python` code package. Regarding Gravitational Waves data generation, we used the `Python` package `PyCBC` to compute GW waveforms, PSD, and noise models, using the approximants *SEOBNRv4* and *TaylorF2*, as described in Chapter 1.

Finally, all our work was run and tested on a regular desktop computer with an Intel core i5-4460 (4 cores, 4 threds) with a 16Gb RAM, 2400 Mhz and using the operating system OS Linux Ubuntu 18.04LTS and no discrete GPUs.

In the following Sections, we provide a detailed description of our method's pipelines.

3.1 The Main Framework

One of the key aspect which distinguishes our work from most applications of machine learning methods in astronomy and cosmology is the fact that we do not provide the training data directly to the regression models.

Instead, we transform it into a compressed basis using PCA and provide the coefficients of that basis, the Principal Components (PCs), as inputs to the models. Since PCA removes redundant information, compressing the data while preserving most of its variance, its application in the supervised machine learning context (PCR) enables the regression models to be much faster while at the same time highly accurate in their predictions. It also enables learning with smaller training sets.

We can breakdown our pipeline into a minimum of six parts. Generation of the initial dataset (including training and test set), employment of dimensionality reduction (on the training data), optimization of the models, final model building (including optimization), estimation of the final quantities and finally evaluation of the models by comparison with the ground truth.

Figure 3.1 shows the general layout of these steps which applies both to the case of Density Field emulation and Gravitational Wave parameter inference.

There are three main aspects of this picture which are distinct in each of our two scientific cases. The first is the initial dataset generation. In one case, we are dealing with N-body simulations of Dark Matter Density Fields and the dataset generation is performed via the Hydra code. In the other, we are

[1] https://github.com/franciscovillaescusa/Pylians3

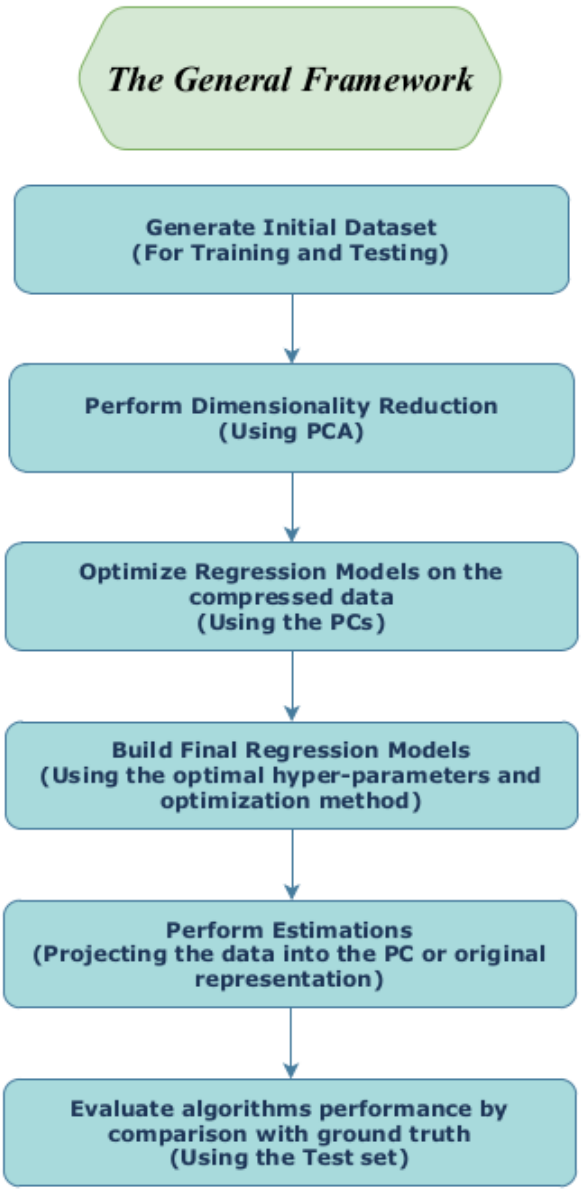

Figure 3.1: Diagram showing the main segments of the pipeline, general to both Density Field and Gravitational Wave pipelines.

dealing with gravitational waveforms, so we generate them via the PyCBC platform and LAL simulation approximants.

The second main difference is in the regressions, the central part of our pipeline. For the case of the density fields, we aim to perform accurate estimations of the 3D matter distribution, given a value for the Dark Matter Density (Ω_{dm}) in our first scenario, and a value for both the Dark Matter Density and the redshift (z), in our second scenario. For the case of the gravitational waves, we aim to infer the Chirp Mass (M_{chirp}) of each test waveform we provide our models both in Time Domain and in Frequency Domain. So it is possible to see that there must be an inversion in the way we build the regression models between the density field problem and the gravitational wave problem, more specifically in the relations between the input and output variables.

In the case of the Density Fields, the dependent (or target) variables for prediction are the PC coefficients while the independent variables are the cosmological parameters. Once we have the PC predictions, we use them to reconstruct the estimated density field by de-projection onto the original data

representation.

Regarding the GWs, our dependent variable will be the Chirp Mass of the GW emitting system while the independent variables will be the PCs obtained from compressing the waveform in the training dataset. Once the training concludes, we project the test waveform dataset on the PC basis provided by the training dataset compression, and the PCs corresponding to each new waveform serve as inputs for parameter estimation.

Finally, the third distinct aspect of the pipeline is how we evaluate the models. In the case of the GW waveforms, since we are dealing with parameter inference, the output of our final estimators will be a set of Chirp Masses, one for each of the test waveforms we provide. After having the estimations, we can compare them with the ground truth values by computing some error metric, in this case, the RMSE.

For the case of the dark matter density fields since our final output will be the density field itself, which translates into a vector of continuous density values we could also compute their RMSE against the vector of the simulated field densities. However, since we are dealing with a cosmological density field we can also compute the power spectra of both estimated and simulated fields, comparing them against each other. This should correspond to a more appropriate evaluation measure since it preserves key statistical properties of the field and it is not too much affected by outliers. Consequently, we will use it as the main evaluator.

We summarize the specificities of each of our scientific cases in Figure 3.2, where we can see the density field pipeline framework in the left flowchart and the GW pipeline on the right. Here we can see and additional distinctive feature in the GW pipeline, where we also have a case of parameter inference from noisy waveforms. To include this experiment we need to expand our pipeline, so we add three additional steps. We compute the power spectrum of the waveforms, generate the noise using the PSD, and finally add it to the strain signal.

To summarize, we so far described the main steps of our work and briefly discussed the main methodology differences in both scientific cases. In the following Sections, we will discuss the details of each part of the Density Fields and GWs pipelines, separately.

3.2 Cosmological Density Fields

Here we describe the first half of our work, where we attempt to estimate N-body dark matter density fields using our proposed compression-based machine learning methods.

In this part we consider two scenarios. In the first scenario, we attempt to estimate the density fields considering just one cosmological parameter as the independent variable used to build the regression models, the Dark Matter Density (Ω_{dm}).

In the second scenario, we extend the pipeline to include the redshift (z) as a second cosmological variable and attempt to use both as independent variables to estimate the density field for different values of Ω_{dm} and at a continuous range of z.

Although the general framework does not change for these two scenarios, there are important details which differ and must be accounted for and that will be commented in the following Subsections.

3.2.1 Training and Test sets: Generation of Density Fields

Here we describe the process of building the train and test dataset of N-body simulations.

To generate this dataset while retaining the bigger picture in mind we need to account for two main factors, the high computational requirements of performing the N-body simulations and that of storing

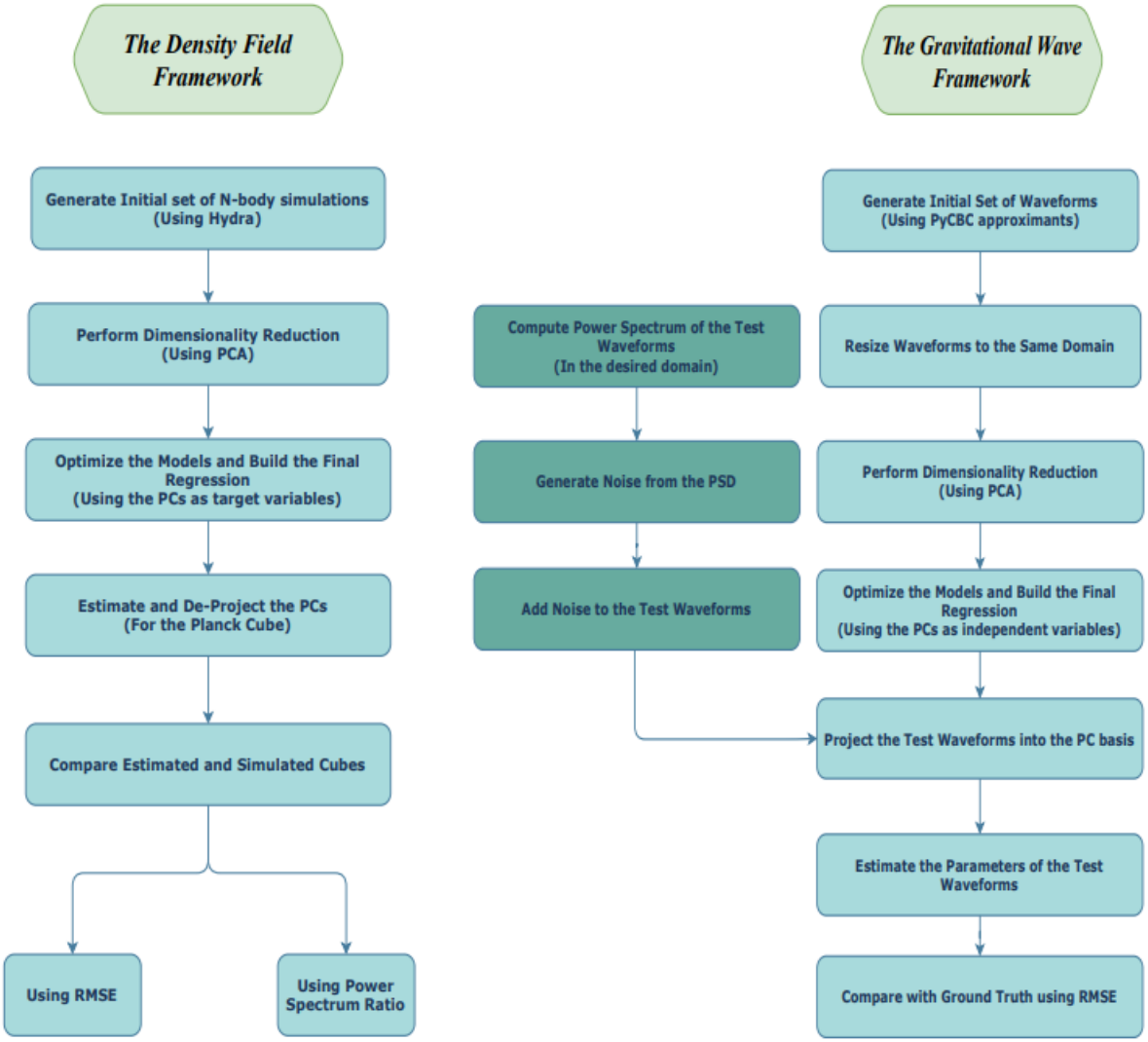

Figure 3.2: Diagrams showing the main segments of the pipeline specific to each scientific case. **Left:** Density Field Emulation. **Right:** Gravitational Waveform Parameter Inference.

and manipulating their outputs. Moreover, these requirements rapidly scale orders of magnitude with the size of the simulations. Given these two factors, it becomes unfeasible to build large sets ($O(10^{3-4})$) of simulations .

Instead, as a demonstration of the validity of our method, we found enough to generate a small set of 24 simulations, setting one of them apart to evaluate the performance of the models.

We generated the simulations using a public version of the Hydra code [19], which solves the Newtonian equations for a set of mass particles of Dark Matter. Every simulation has the same initial conditions, starting from the first snapshot at $z = 49$ until reaching $z = 0$, summing up to a total of 10 snapshots. We generated the fields for 160^3 particles of dark matter initially positioned on a regular cubic grid of comoving length $L = 100\text{h}^{-1}$Mpc on the side, and every snapshot contains positions and velocities for each one of them.

To determine the transfer function of the matter power spectrum the BBKS formula [55] was used with a shape parameter $\Gamma = \Omega_m h$ [56].

The cosmological model assumed was the Λ-*CDM* model with $\Omega_k = 0, \Omega_m = 1 - \Omega_\Lambda, \sigma_8 = 0.809$, and $h = 0.7$, where σ_8 corresponds to the Root Mean Square Error (RMSE) of the density fluctuations when convoluted with a Kernel at a scale of $8\ \text{h}^{-1}$Mpc and h is the reduced Hubble constant (i.e. $\text{h} = \text{H}_0/100$).

Regarding the training set, the 23 simulations were generated with Ω_m varying in the interval $\Omega_m \in [0.05, 0.6]$ with $\Delta\Omega_m = 0.025$. Our test simulation was generated with the density value from the Planck collaboration 2015 [57], corresponding to $\Omega_m = 0.309$. With this choice of parameters the mass of the dark matter particles is given by $m_{dm} = \Omega_m \rho_{crit} V$ where V is the volume of the box and ρ_{crit} is the critical density given by $\rho_{crit} = \frac{H^2}{8\pi G}$.

After calculating the positions and velocities for each particle using the Hydra code, the data is compiled by another code called *darkdens* which calculates the densities of each one of the particles using the *Smooth Particle Hydrodynamics* (SPH) formalism [58].

Finally, the data goes through yet another fundamental piece code which organizes the particles in 128^3 cubic voxels, calculating the density in each one, which has dimensions of $10^{27}\ Kg\ m^{-3}$ per voxel.

Regarding the scenario of multiple parameter (Ω, z) density field estimation, there is no need to perform additional N-body simulations. During the production of the simulations, we stored ten snapshots are different redshifts. Since each of the 24 performed simulations already includes the ten redshift snapshots we have 24×10 density fields, each Ω_m cube spanning ten redshifts. The redshift range corresponds to $z = [0, 0.25, 0.5, 0.75, 1, 1.5, 2, 3, 5, 10]$.

Unfortunately, our computational resources do not enable us to use the full data set on our pipeline. Both the compression and the regression steps have memory requirements exceeding those available to us if we use the total dataset. Consequently, we decided to use only four redshifts for each one of the 23 density fields in the training set.

Thus, regarding the z and Ω_m density field estimation scenario, the training set that we adopted in this work comprises 23 Ω_m density fields spanning four redshifts, $z = [0, 0.25, 0.75, 1]$, while $z = 0.5$ for $\Omega_m = 0.309$ was held for testing. Consequently, the final training dataset will consist of a total of 92 density cubes.

In Figure 3.3 we show three 3D colour plots of the resulting density fields for increasing Ω_m where it is possible to notice the effect of varying the average density by looking at the filaments which gradually start to become more pronounced. We can also observe, looking at the colour map, that the densities in the void regions also increase for higher values of Ω_m. These plots were obtained using the function

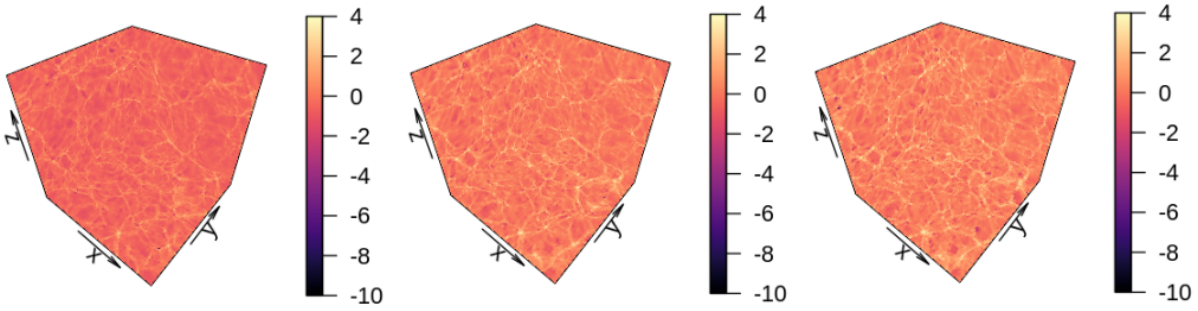

Figure 3.3: 3D Color Plots of three Simulated Density Cubes for increasing Ω_m and at $z = 0$. **From left to right:** $\Omega_{dm} = 0.05$; $\Omega_{dm} = 0.3$; $\Omega_{dm} = 0.6$. **Units:** $\left(10^{27} \frac{Kg}{m^3}\right)$

`slice3D` from the R CRAN package `plot3D` [59].

3.2.2 PCR for Density Field Estimation

Estimation on Ω_m

After performing the N-body simulations, we store our data in two separate directories. One with 23 files, the training set, where each file contains 128^3 values for the density cell, and another with one text file, corresponding to the 128^3 values for the density cells in the Planck density field that would be used as the testing cube..

We then proceed to load our training set into a matrix. Each row corresponds to a density field built upon a specific Ω_m value, and each column corresponds to a density cell value. After loading our training set into the *R* environment, we end up with a matrix of 23 observations spread across $128^3 = 2097152$ dimensions. Each observation corresponding to a density field associated with a different Ω_m value. Finally, we add an additional column with the Ω_m values, which will be the vector of independent variables in our regression models.

The next step in our pipeline is to perform dimensionality reduction using PCA. For that purpose we use the function `prcomp` from the package `stats`, with the parameters `scale` and `center` set as `TRUE`, and apply it to our training matrix in its logarithmic form, excluding the column of Ω values. This function does not perform the regular PCA, which would imply an eigendecomposition of our covariance matrix. Instead, it performs *Singular Value Decomposition* (SVD), which automatically decomposes an arbitrary matrix into the three explained components. It is possible to show that an SVD performed on centred data reduces to the standard PCA via eigendecomposition of the covariance matrix. This way, by setting the parameter `center=TRUE` we are ensuring the application of a standard PCA. The actual centring procedure corresponds simply in subtracting the columns means to each element in the respective column. Additionally, by setting `scale=TRUE` the function will also scale the data before applying the SVD procedure. Meaning that each (already centred) column in the data matrix will be divided by its standard deviation, ensuring that all the variables have the same variance and thus removing the bias towards higher variances.

After computing SVD, the function will return five objects. A vector `sdev` containing the standard deviations of each Principal Component. Two vectors `center` and `scale`, containing respectively the values used in the centring and scaling of our original data matrix. The matrix `rotation` containing the *loadings* (in each column), holding the correlations between the original data and the PCs. Finally, we have the scores matrix `x`, which contains the PC scores, or in other words, our data projected in the principal directions. If we consider the loading and scores matrices to be given by **L** and **S**, the scale and centre vectors to be given by **s** and **c**, to obtain our original data representation the scores must be matrix multiplied by the loadings with a posterior sum of the centre vector and multiplication of the scaling one, yielding:

$$\boldsymbol{M_{orig}} = 10^{(\boldsymbol{SL^T s^T + c^T})} \tag{3.2.1}$$

Once we have our projected quantities, we build and optimize the regression models. Considering that the optimization process is lengthy, involving lots of relevant parameters, different schemes and since it is intimately related to how we evaluate the models, it will be described in detail in Subsection 3.2.3, together with the evaluation metrics. Here we only describe the way we build the final density cube estimator.

Since each density field is represented by 23 PCs, and we aim to estimate the field itself, the training is performed using the PCs as dependent variables and Ω_m's as independent ones. Once trained, they are used to predict the new set of 23 PCs for $\Omega_m = 0.309$, which can be de-projected into the final estimated

cube. To achieve this, we create a 23 iteration loop wherein each iteration our four different algorithms build a regression model accounting for that specific PC and the same constant set of Ω_m values.

Depending on the algorithm we are using, different R functions are applied which may receive the input data in different ways. Regarding the algorithms themselves for the neural networks, random forests, extremely randomized trees and support vector machines our choice was to use, respectively, the functions `nnet`, `randomForest`, `extraTrees` and `svm` from the R packages `nnet` [60], `randomForest`[61], `extraTrees` [62] and `e1701` [42]. All the algorithms receive their inputs in the form `x=predictor` (vector of PCs) and `y=target` (vector of Ω_m's), the variables can be either in matrix/vector form or in a data frame, except for the case of `extraTrees` where it is mandatory to give the inputs in matrix/vector form.

For the case of the neural networks, we provide the data normalized to the [0,1] range, since it can diminish the chances of getting stuck in local minima. To do that we apply min-max normalization to each column **C** of our matrix **S** containing the PC scores, yielding:

$$\boldsymbol{C_{norm}} = \frac{\boldsymbol{C} - \min(\boldsymbol{C})}{\max(\boldsymbol{C}) - \min(\boldsymbol{C})} \tag{3.2.2}$$

where $\boldsymbol{C_{norm}}$ corresponds to the normalized column.

After optimizing the hyper-parameters in the same loop described previously, the 23 final regressions are built and stored on a list.

To obtain the final PC predictions using these models, we rely on the function `predict` from the R package `stats`. This function accepts as inputs the regression model and the predictor variables, which in this case correspond to one of the four algorithms and $\Omega_m = 0.309$, respectively. Consequently, to estimate the 23 PCs, another 23 iteration loop is built where we use this function and our 23×4 models to continuously predict the 23 new PCs for the same Ω_m value.

After having the new PCs, the four final estimated cubes are obtained, by de-projecting them on to the original representation using equation 3.2.1.

In the case of neural networks, we have to first de-normalize the data to the original range using:

$$\boldsymbol{PC_{denorm}} = \boldsymbol{PC_i}(\max(\boldsymbol{C_i}) - \min(\boldsymbol{C_i})) + \min(\boldsymbol{C_i}) \tag{3.2.3}$$

where $\boldsymbol{PC_{denorm}}$ corresponds to the de-normalized PC representation, $\boldsymbol{PC_i}$ to the ith estimated PC score and $\boldsymbol{C_i}$ the corresponding ith column of the **S** matrix.

Once we have our final estimated cubes, we can proceed and evaluate the models. However, before that, we will explain how we deal with the scenario of the two-parameter based estimation.

Estimation on Ω_m and z

Regarding the scenario of estimation using two-parameters, the first main difference concerns the training data set. To incorporate the training in redshift space considering the computational resources available, we decided to use four redshifts, $z = [0, 0.25, 0.75, 1]$, maintaining our 23 Ω_m values for each of those redshifts. For the sake of evaluating the models, we chose our test sample to be the same $\Omega_m = 0.309$ simulation, but at redshift $z = 0.5$, which wasn't included in the training datase..

The whole pipeline takes significantly longer to run since instead of having 23 simulations as it was the case for the single parameter estimator, compressed to a representation of 23 PCs, we have $23 \times 4 = 92$ simulations, compressed in a basis of 92 PCs. Considering this situation, instead of building a loop of 23 iterations, we have 92 iterations, where in each iteration we once again focus on a single PC vector,

providing it as the dependent variable. Regarding the independent variables, we now provide a 2 column matrix with both Ω_m and the redshift values, in the first and second columns, respectively.

After building the 92 models, we once again store them in a list. Afterwards, this list serves as input to the `predict` function in another 92 iteration loop, where now we provide both $\Omega_m = 0.309$ and $z = 0.5$ to obtain the respective PCs.

The de-projection works in the same way as explained in the first scenario, following equation 3.2.1.

3.2.3 Optimization and Evaluation of the Models

Having the core part of our pipeline explained, we will now provide the details on the optimization and evaluation of the algorithms.

As previously mentioned, this is the most time-consuming segment of our pipeline. That is because we not only optimize the algorithms with a given scheme and range of hyper-parameters, but we also try different optimization schemes (e.g. Cross-Validation, Bootstrap) with different hyper-parameter ranges. So in a way, we are also trying to select the hyper-parameter optimization process that gives the best result for the combination between the data and the machine learning method, and that naturally demands a high amount of computational resources.

To apply the optimization schemes, we use two different functions depending on the machine learning method that we are optimizing. The `tune` function from the `e1071` R package [42] is used for Random Forest and SVM, and the `train` function from the `caret` package [63] for the Extremely Randomized Trees and Neural Networks.

Optimization Packages Specificities

Regarding the `train` function, our application specifies the following parameters, `train(form, data, method, tuneGrid, trControl, numThreads)`. In `form`, we provide an expression defining which set of parameters is the predictor (Ω_m's or (Ω_{dm},z)) and which set is the target variable (PCs). This formula depends on the data frame we provide in the parameter `data` which must have a column for the target variables and a column (or more for higher dimensions) for the predictors. In the first scenario, since in each iteration we focus on a single parameter (Ω_{dm}), we are dealing with one-dimensional regressions, and thus our data frames will have just two columns, one regarding the dark matter densities and the other regarding the corresponding PC vector. Consequently, if the density column is called "omega" and the PC column is called "PC", the formula can be given as `form=PC~omega`. In the case of the two-parameter based estimation, we are dealing with one more dimension in the independent variable space, and thus, we will have a three-column data frame, with an additional redshift column. Consequently, if we call that column "redshift", we can provide the formula as `form=PC~omega+redshift`, or simply `form=PC~.`, where this latter notation applies to an arbitrary number of independent variables, provided that we include them in the data frame given in the `data` parameter.

The parameter `method` specifies the regression model we will use, so we simply provide the name of the function we build the final regression models with, `method = 'nnet'` and `method = 'extraTrees'`, for the neural networks and extremely randomized trees, respectively.

The `tuneGrid` is a previously defined grid with the hyper-parameters we want to optimize and the respective ranges, so we use the `expand.grid` function from the `base` package.

The `trControl` parameter consists in the control structure defining the optimization scheme and details related to it. For that purpose we use the the function `trainControl`, with the parameters `method`

and `number` specifying the optimization schemes we want to use, `method='boot'` with `number=N` for bootstrap with N repeats, and `method='cv'` with `number=K` for K-fold cross-validation.

Finally, the `numThreads` specifies the number of threads we want to use to compute the models.

In the case of the `tune` function we have the same parameters, a formula with the same syntax, a data frame provided in the `data` parameter, and the method provided using `method='svm'` for the support vector machines and `method='rf'` for the random forest.

The control structure defining the optimization scheme is provided to the parameter `tunecontrol` using the function `tune.control`, with `sampling='boot'` and `nboot=N` for bootstrap with N repetitions and `sampling='cross'` with `cross=K` for K-fold cross validation.

Finally, the hyper-parameters and their ranges do not need specification through a grid, they are instead provided directly as parameters of the `tune` function, and can also be defined as vectors through the use of the command `seq`.

Now, to find the optimal model in the range of hyper-parameter values we define, the function `train` uses the *Root Mean Squared Error (RMSE)*, defined as:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{3.2.4}$$

where $\hat{y}_i$ is the ith estimated PC value for a model built with a certain choice of hyper-parameters, y_i is the actual PC value and n is the size of the considered sub-sample of the training set.

The function `tune` uses instead the Mean Squared Error (MSE) which is equivalent to 3.2.4 without the square root.

General Optimization Process

In this Subsection we explain the optimization methdology that was adopted in this work.

The described optimization functions run internally to our PC regression building loop. Once we fix the hyper-parameter range, as well as the optimization scheme, they remain constant throughout all PC iterations. In each iteration, the R functions search through the fixed hyper-parameter range and find the optimal hyperparameters as the ones giving the least RMSE (or MSE) on a sub-sample of the provided training dataset. Each different hyper-parameter choice will have the regression fit in a different sub-sample of the training set. After all the iterations conclude, we have a set of regression models, each optimized to estimate a single Principle Component, for the chosen scheme and hyper-parameter range. Having those regressions built, we can individually estimate each PC for the new density field we want to emulate, and finally de-project the estimated PCs into the original density field representation. Afterwards, we repeat the process for a different range of hyper-parameters, using again the best-found regression models to predict the new set of PCs for de-projection.

Once we search all of the desired ranges, we have a set of density fields, one for each tried range, available to be compared against the ground truth, in what we call the external optimization process.

To implement the external optimization we need to define an evaluation measure, which effectively compares the final estimated density field to the simulated one. For that purpose, we define two evaluation metrics. One we call the *Mean Over-density Distance* (MOD), defined as:

$$MOD = \sqrt{\frac{\sum_{i=1}^{N} \left(\frac{\rho_i^{sim}}{\bar{\rho}^{sim}} - \frac{\rho_i^{est}}{\bar{\rho}^{est}} \right)^2}{N^2}} \tag{3.2.5}$$

where ρ_i^{sim} and $\bar{\rho}^{sim}$ correspond to the density value in the *ith* pixel and the mean density in the simulated density field, respectively, ρ_i^{est} and $\bar{\rho}^{est}$ correspond to the density value in the *ith* pixel and mean density of the estimated density field, and finally N corresponds to the number of pixels in the simulated field.

As the name implies, this is a measure of the distance between over-densities and should reflect how close the density pixels are to each other in both estimated and simulated fields.

The other measure we call the *Power Spectrum Ratio Distance* (PSD_{rd}) and is defined as:

$$PSD_{rd} = \sum_{i=1}^{n} \left| 1 - \frac{y_{est}^i}{y_{sim}^i} \right| \tag{3.2.6}$$

where y_{est}^i is the *ith* cell in the power spectrum vector of the estimated field, y_{sim}^i regards the *ith* cell of the simulated one and n is the size of the power spectrum vector.

As explained before, this quantity gives a complete description of the statistical properties of the density fields. Contrarily to the RMSE, the power spectrum shouldn't be dependent on the initial seed of the simulations. Consequently, it should give a more appropriate error measurement for this case of random density fields. Of course, since we want the estimations to be as statistically close as possible to the simulations, this quantity needs to be as close to zero as possible.

Now, since each algorithm has different sets of relevant hyper-parameters, and many have an infinite range of possible values, we first try a small set of search ranges for a fixed optimization scheme, which we chose to be 10-fold Cross-Validation. This choice for 10-fold CV stems from the fact that this scheme is generally quite successful in capturing most of the variance in the dataset while maintaining minimal bias, while at the same time is not highly computationally demanding.

After trying a given set of hyper-parameter range searches for the mentioned choice of optimization scheme, we evaluate each final model by computing both the RMSE and the Power Spectrum Ratio.

Having our optimal hyper-parameter search range found, we try that same range with different optimization schemes. Our choices were 8-fold CV, 12-fold CV, 23-fold CV and finally, 20-repeat bootstrap. After evaluating these schemes, we build the final model with the one corresponding to a PSD_{rd} as close to zero as possible.

Now, we will discuss the specifics in the optimization of each algorithm.

Random Forest Optimization

As explained in Chapter 2, the hyper-parameters we chose to optimize regarding this algorithm were the number of trees in the ensemble, the minimum size of the terminal nodes and the number of features randomly considered in each split. Regarding the `randomForest` function, these correspond respectively to the parameters `ntree`, `nodesize` and `mtry`.

We chose to use the `tune` function to optimize Random Forest since it enables the optimization of all of the three previously mentioned hyper-parameters, while the `train` function only enables the optimization of `mtry`.

One important fact to mention, is that since the parameter `mtry` reflects the number of features we are considering for each tree split, it will only be relevant to optimize it when we have more than one feature in our training dataset, corresponding to the case of Ω_m and z estimation. Consequently, for the single Ω_m estimation, the only parameters that we will be dealing with are `ntree` and `nodesize`.

For both cases, we noticed that sometimes optimizing two hyper-parameters together gave worse results than optimizing just one, fixing the optimal value and optimizing the other afterwards. Given that, our approach to optimize random forest was to start by doing a simple grid search over three different

tree number values, `ntree=(500,1000,2000)`. After finding the optimal `ntree` value, we fix it and optimize the remaining hyper-parameters.

In the case of Ω_m estimation, we optimize the `nodesize` parameter in the range `nodesize=[1;15]` with Δ`nodesize=1`, using the four chosen cross-validation schemes and bootstrap and building the final model with the one giving the smallest PSD. So we repeatidly try the same range of optimization, with different optimization approaches.

For the case of Ω_m and z estimation, we also need to optimize the parameter `mtry`. After findig the optimal number of trees through Grid-Search, we fix it and use 10-fold CV to optimize `mtry` with the values `mtry=(1,2)` and `nodesize` in the same range as previously, both simultaneously searched. Then we try to optimize each hyper-parameter individually with the remaining one set to its default value. For this case, given the best `mtry` search sequence found in the individual `mtry` search, we also do an additional experiment in extending the `nodesize` range to `nodesize=[1;30]` with Δ`nodesize=1` since we have four times more observations than in the single feature case. Finally, given the best hyper-parameter search combination found, we apply that same combination with the remaining optimization schemes, to attemp to further decrease the regression errors.

Extremely Randomized Trees Optimization

For this case, as explained previously, we are going to optimize the number of trees, the number of random cuts and the number of randomly selected features in each split. Regarding the `extraTrees` function, these correspond respectively to the parameters `ntree`, `numRandomCuts` and `mtry`.

We chose to use the `train` function since currently, the `tune` function does not enable the optimization of `extraTrees`. Additionally, we are not optimizing the `nodesize` since the `train` function does not enable it for this algorithm.

Regarding the optimization process itself, it is almost identical to the random forest case, if we substitute the `nodesize` parameter with the `numRandomCuts`.

For the case of one-parameter regression, we start with the same `ntree` grid-search as in Random Forest, fix the optimal value, and run the remaining optimization schemes on `numRandomCuts=[1;15]` with Δ`numRandomCuts=1`.

For the case of two-parameter estimation, after fixing the tree number, we try the same simultaneous and individual optimization process for both `mtry` and `numRandomCuts`. The difference being the fact that for this case we search in a larger range, `numRandomCuts=[1,25]` with Δ`numRandomCuts=1` and after fixing the optimal `mtry` (found in the individual search) we make an additional extended search over the range `numRandomCuts=[1;40]` with Δ`numRandomCuts=1`. The reason why we extend the ranges to larger intervals is simply the fact that `extraTrees` is quite faster than `randomForest`, so we can afford to increase the memory requirements.

After finding the best configuration from the previous cases, we once again fix it and run the remaining optimization schemes.

Neural Networks Optimization

Regarding the neural networks, we choose to optimize the number of neurons in the (single) hidden layer and the weight decay parameter. Regarding the `nnet` function these are respectively, the parameters `size` and `decay`.

For this case, we could have chosen to optimize through the `tune` function, since not only enables the optimization of both hyper-parameters but is also faster than `train`. We chose the `train` function

through experimenting on both performances for the same range of hyper-parameter values. We chose to perform an optimization of the hyper-parameter `size` in the range `size=[1;25]` with Δ`size=1` with both the `train` and the `tune` functions, while fixing the rest of the hyper-parameters to their default values. The outcome of the experiment gave better PSD results for the `train` function, and thus we chose it as our optimization model.

Regarding the optimization process, we once again try both individual and simultaneous optimizations. In order to do that we first applied 10-fold CV to the hyper-parameter `size` in the just mentioned range and to `decay` in the range `decay=[0,1]` with Δ`decay=0.1`, in a simultaneous search. Afterwards, we fixed `decay` to its default value `decay=0`, optimized the `size` parameter individually. This time it is not possible to do the reverse (fixing `size` to default and individually optimizing `decay`), since the `size` does not have a default value. Interestingly, for each redshift, every optimal result consisted of the one where we fixed `decay` to the default value.

Given that knowledge, we decided to remain with the default `decay` value and only optimize `size` in different ranges, while still applying the 10-fold CV scheme. The ranges we chose were `size=`$\{$`[1;25],[1;50],[1;75],[1;100]`$\}$, and after running 10-fold CV in each one of these ranges, the best performing one was chosen for the remaining optimization schemes searches.

For the case of two-parameter estimation we did a similar process but including only the `size=`$\{$`[1;25],[1;50]`$\}$ ranges, since this scenario implies a much more computationally heavy pipeline.

Support Vector Machine Optimization

Here, the parameters we chose to optimize were the Kernel of the support vector machine and its associated relevant parameters, γ and r, as explained in Chapter 2. Concerning the `svm` function these are, respectively, the hyper-parameters `kernel`, `gamma` and `cost`.

We decided to use the `tune` function since is tailored to tune this algorithm and it belongs to the same package, so it is much faster than the `train` function.

Regarding the optimization process, we first tried to find the optimal kernel. We achieved that by building models with the four different possible kernels (linear, polynomial, sigmoid and radial basis) while setting the remaining hyper-parameters to their default values, and choosing the one which yields the lowest PSD. We found that the radial kernel easily outperforms the remaining ones.

Afterwards we used 10-fold CV to optimize the `gamma` and `cost` hyper-parameters. We tried two different ranges, considering the intervals $I_1 = [-5; 3]$ with $\Delta I_1 = 1$ and $I_2 = [-10; 4]$ with $\Delta I_2 = 2$ our ranges were defined as $R_1 = 10^{I_1}$ and $R_2 = 10^{I_2}$.

Then, for each range, we experimented optimizing the two hyper-parameters both simultaneously and individually. After obtaining the best configuration, we applied the remaining optimization schemes to it.

For the case of two-parameter estimation, there were no changes in the optimization process.

3.2.4 Compressing the Pipeline through FPCA

As explained in Chapter 2, we also attempt to make the whole pipeline more compact, by fusing the dimensionality reduction features of PCA with the predictive ones from our supervised machine learning algorithms through the use of Functional Principal Component Analysis.

To apply this method, we use the R package `fda` [64] in a process divided into three steps.

The first step is to build our *Functional Data Object* from the training data. To do that we first need to specify the set of basis functions used to compute the FPCA. To go in accordance with our application

of PCA, we defined 23 b-spline basis functions, using the function `create.bspline.basis`. In this function, we need to provide a vector defining the interval where our functional data would be evaluated. This vector is provided through the parameter `rangeval`, and will correspond to the range of Ω_m values ($\Omega_m = [0.05; 0.6]$, $\Delta\Omega_m = 0.025$). Finally, the number of basis is defined in the parameter `nbasis`.

After having the basis functions, we need to provide them together with the training data and our vector of argument values to the `Data2fd` function, which will convert the data into the final `Functional Data Object` where the harmonics/FPCA's will be evaluated. This function accepts the training data in the same way as in the PCA case, a matrix where each row corresponds to a density cube in vector form, and with previously applied logarithmic scaling. This matrix, the range of Ω_m values and the set of basis, are provided in the parameters `y`, `argvals` and `basisobj`, respectively.

The second step consists in applying the function `pca.fd` to our freshly created functional data object to compute its harmonics, providing it in the parameter `fdobj` and defining the number of harmonics by setting `nharm=23`. This function returns five objects. The objects `harmonics` and `values` contain, respectively, the eigenfunctions and eigenvalues. The object `scores` corresponds to the matrix of scores used to de-project the data, `varprop` is a vector with the proportion of the variance explained by each eigenfunction, and finally `meanfd` gives the mean function used to center the data. Here, we do not center the data, but it is possible to do it by setting `centerfns=TRUE`.

The last step consists of estimating the new PCs for the test Ω_m we desire ($\Omega_m = 0.309$), and de-project them into the estimated cube. Since we already have our harmonics, which correspond to a continuous form of the Principal Components, where each continuously varies with Ω_m, we need to provide the object with the computed harmonics and the Ω_m we want to predict in the parameters `fdobj` and `evalarg` of the `eval.fd` function.

Finally, since we are not forcing the FPCA method to center and scale our data, the de-projection does not involve scaling and centering vectors like in equation 3.2.1. We only need to multiply each new PC obtained in the previous step by each column of the matrix `scores` returned by the `pca.fd` function.

One last important remark is the fact that due to the computationally expensive nature of this method our resources only manage to apply it to simulations of 64^3 density cells, instead of the 128^3 density cells that we are dealing with in all the other cases.

3.3 Gravitational Waves

Finally, we reach the second and final part of our work. In this Section, we describe the methodology associated with the Gravitational Wave parameter inference. As in the cosmological density fields, this part of the work comprises two scenarios.

In the first scenario, we work with waveforms in Time-Domain (TD) while in the second, we work with waveforms in Frequency Domain (FD). In both cases, we will be training the algorithms using waveforms without noise added and testing the algorithms by making the inferences in a dataset of waveforms imbued with noise, with decreasing S/N ratios.

In the following Subsections, we will be describing the process of waveform and noise generation, how we implement the pipeline for GW parameter inference and finally, the process of optimizing the algorithms and evaluating their performances.

3.3.1 Waveform and Noise Generation

Waveform Generation

We implement the process of waveform and noise generation through the Python PyCBC platform. For both TD and FD scenarios, we generate a training set of 256 waveforms, varying their chirp mass in the range $M_{chirp} = [9.13; 25.80]M_{\odot}$, and a test set of 255 waveforms with their chirp mass in the range $M_{chirp} = [9.19; 21.79]M_{\odot}$. Keep in mind, that although in order to avoid extrapolation the ranges overlap, there is no training waveform with the same chirp mass as a test waveform. To achieve that, for the training set we fix one of the GW sources mass as $M_1 = 10$ and vary the second source mass in the range $M_2 = [11, 74.75]$ with intervals of $\Delta M_2 = 0.25$. For the test set we also fix one of the GW sources mass as $M_1' = 10$ and vary the second source mass in the range $M_2' = [11.15; 74.65]$ with intervals of $\Delta M_2' = 0.25$. This way, we ensure that there is no degeneracy between both sets.

For the case of Time Domain, we use the function `get_td_waveform` from the `pycbc.waveform` package, providing the masses in the parameters `mass1` and `mass2`. Of course, since each function only provides one waveform at a time, we need to build a loop, iterating the second mass in the mentioned range. There is also a need to provide the time resolution of the waveform (Δt) and a lower frequency limit (f_{low}). For the former we set it to $\Delta t = \frac{1}{4096}$ using the parameter `delta_t`, for the later and in order to go along the lines of the LIGO project, we set it to $f_{low} = 10\ Hz$, using the parameter `f_lower`.

One crucial aspect of our work is the need to provide a distance, which will be our tool to tweak the S/N ratio for inferences with noise. We will be making inferences for the same test set, with algorithms trained on the same waveforms at varying distances (eg. 1 Mpc, 25 Mpc, 50 Mpc, 100 Mpc, 500 Mpc and 1000 Mpc). To achieve that we simply provide the distance in the parameter `distance`.

Finally, we also need to provide an accurate approximator to compute the observed waveforms generated by a system modelled by the defined parameters. To achieve that we use the *SEOBNRv4_opt* approximant, providing this string in the parameter `approximant`.

For the case of Frequency Domain we use the function `get_fd_waveform` from the same package, providing the masses, distance and low-frequency cutoff in the same way as in the case of TD waveform generation. The main differences reside in the need to provide a frequency resolution (Δf), a higher frequency cutoff (f_{high}) and a frequency domain approximator.

For the higher frequency cutoff, once again, to go along the lines of LIGO we set it to $f_{high} = 10\ kHz$, providing it in the parameter `f_higher`. Keep in mind that this does not mean that our waveforms will be reaching these frequencies since the duration of the collapse depends on the chirp mass of the system.

We provided the frequency resolution in the parameter `delta_f` , setting it to $\Delta f = \frac{1}{16}$, and finally we choose the approximant *TaylorF2* to generate the waveforms.

In Figure 3.4 we can see a comparison between a Time Domain waveform and a Frequency Domain one, both taken from our training dataset before the PCA projection and both with chirp masses corresponding to $M_{chirp} = 17.1442$.

Looking at this image, the difference in the generated waveforms resolution for these two contexts becomes clear.

Regarding the physics themselves, we can easily see, at least looking at the Time Domain picture, both the inspiral and merge phases. These phases are followed by a barely visible ringdown, which terminates abruptly due to our process of cropping the waveform zeros.

After running the training and test set loops, we store the waveforms in a list in order to make the posterior noise addition. The generation of such noise will now be explained.

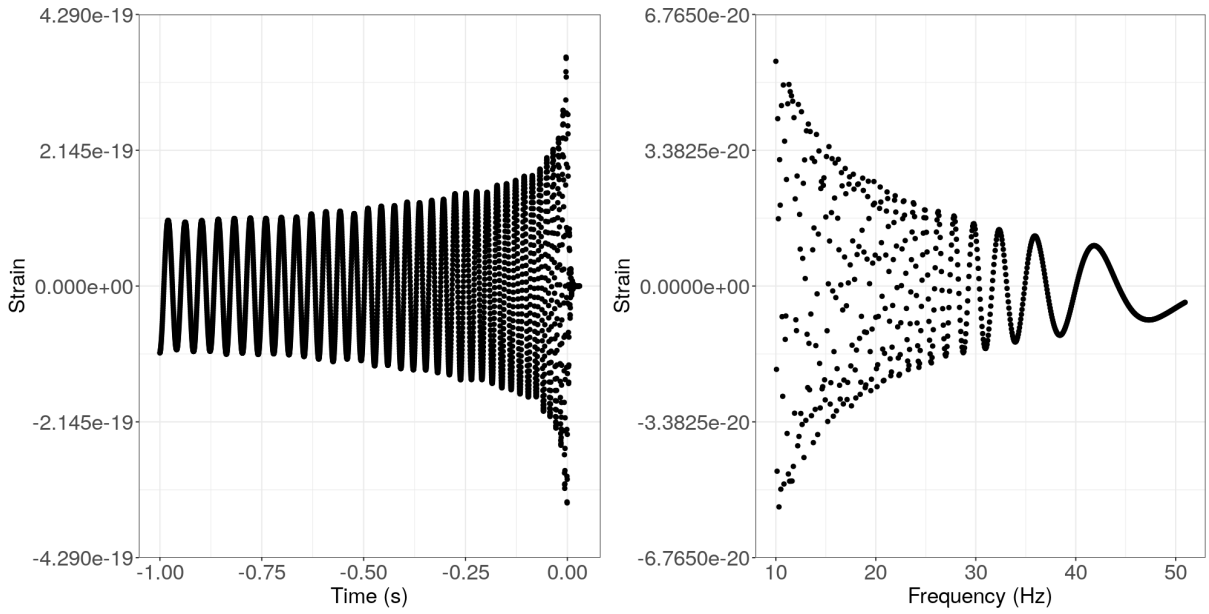

Figure 3.4: Comparison between Time Domain (left) and Frequency Domain (right) waveforms with $M_{chirp} = 17.14\ M_{\odot}$, taken from the Training Dataset.

Noise Generation

To colour the noise, we need to generate an analytical power spectral distribution (PSD). To achieve that, we use the function `pycbc.psd.analytical.aLIGOZeroDetHighPower` from the package `pycbc.psd`, which generates an analytical Gaussian Noise simulation expected for the Advanced Ligo detector, where the detector is assumed to be in zero detuning and high power. This function accepts three quantities, the length of the PSD (l_{PSD}), the frequency resolution (Δf) and the low-frequency cutoff (f_{low}).

For the case of Time Domain the length of the PSD is provided as $l_{PSD} = \frac{2048}{\Delta f} + 1$ in the parameter `length`, where we included the fundamental mode in the picture, the frequency resolution was defined as $\Delta f = \frac{1}{16}$ in the parameter `delta_f`, and finally we set the low frequency cutoff in the same way as in the waveform generation, $f_{low} = 10\ Hz$, in the parameter `low_freq_cutoff`. After obtaining the PSD, we use the function `pycbc.noise.gaussian.noise_from_psd` from the `pycbc.noise` package. We provide four parameters to this function. The PSD itself (*psd*), obtained through the mentioned function. The length of the noise (l_{noise}), corresponding to the number of points in the generated waveform. And finally, the time resolution (Δt), which will be, once again, matched to the time resolution of the generated waveform $\left(\Delta t = \frac{1}{4096}\right)$.

For the case of Frequency Domain the function which generates the noise only accepts the PSD as an input, since it already specifies the length and frequency resolution. Given this fact, we provide the length of the waveform in the PSD length parameter and the frequency resolution of the waveform ($\Delta f = \frac{1}{16}$) for the frequency resolution parameter. In order to obtain the noise we use the function `pycbc.noise.gaussian.frequency_noise_from_psd` from the `pycbc.noise` package, providing the previously defined PSD.

For both cases, after obtaining the noise vector, it is just a matter of adding it to the strain obtained through the waveform generation function.

In Figures 3.5 and 3.6 we can see a visualization of the waveforms after being added with the noise vectors and at increasing distances, for time domain and frequency domain waveforms, respectively.

There are two things worth mentioning in these pictures.

One is that for our choice of PSD, the obtained noise only becomes relevant for distances nearing

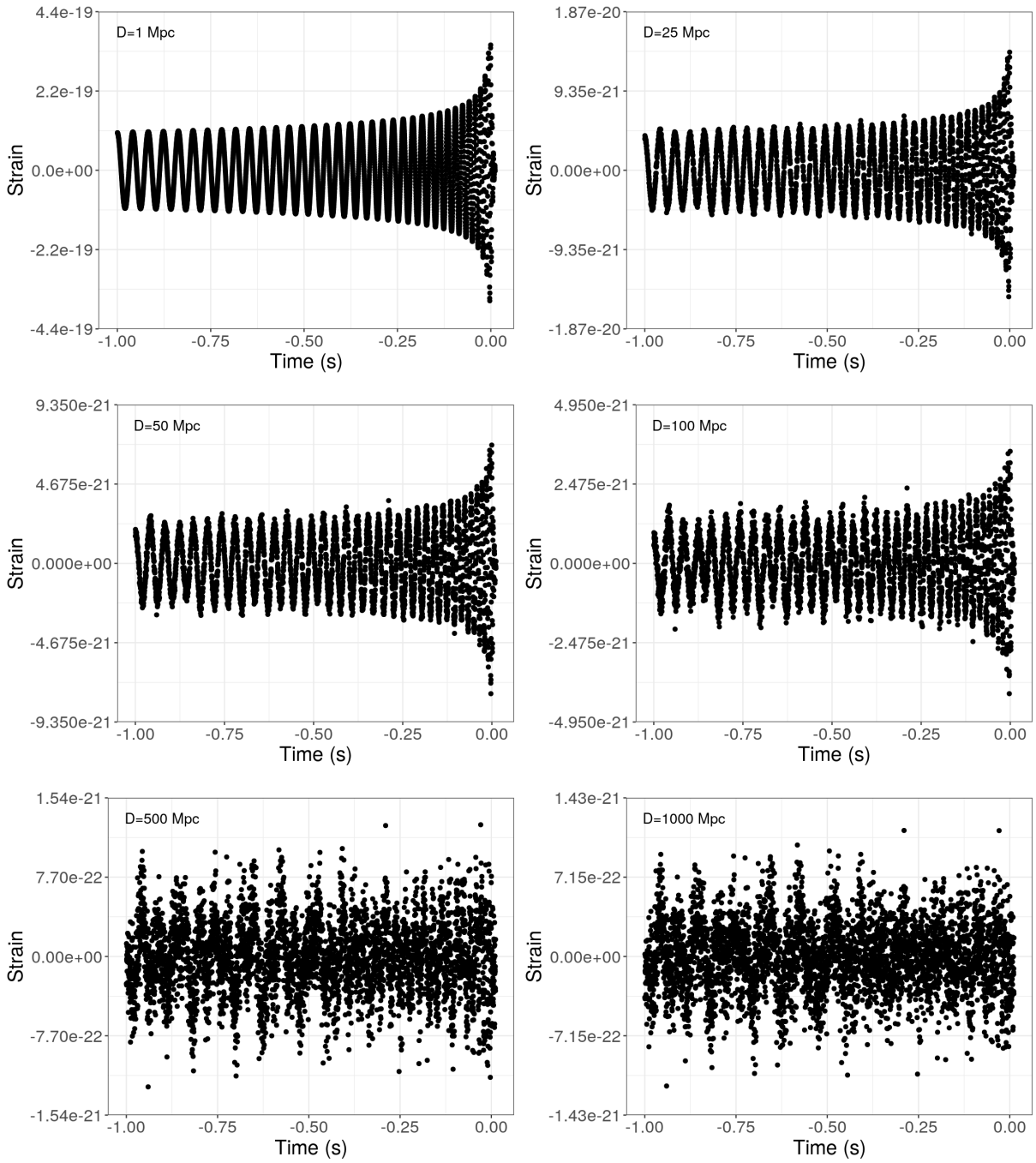

Figure 3.5: Visualization of six Time Domain Waveforms with $M_{chirp} = 17.17\ M_{\odot}$ at increasing distances (decreasing signal-to-noise ratios), taken from our test dataset.

D=25 Mpc, with D=1 Mpc being very similar to the case of a waveform without noise. In other words, the S/N ratio only falls to considerably low values around the mentioned distance.

Secondly, the TD waveforms shape becomes quite more suppressed at extremely low S/N ratios than the FD waveforms shape. This stems from the fact that these waves have a smoother distribution over all of the time domain, which gets rapidly diluted in the noise as the distance/strain amplitude increase/decrease.

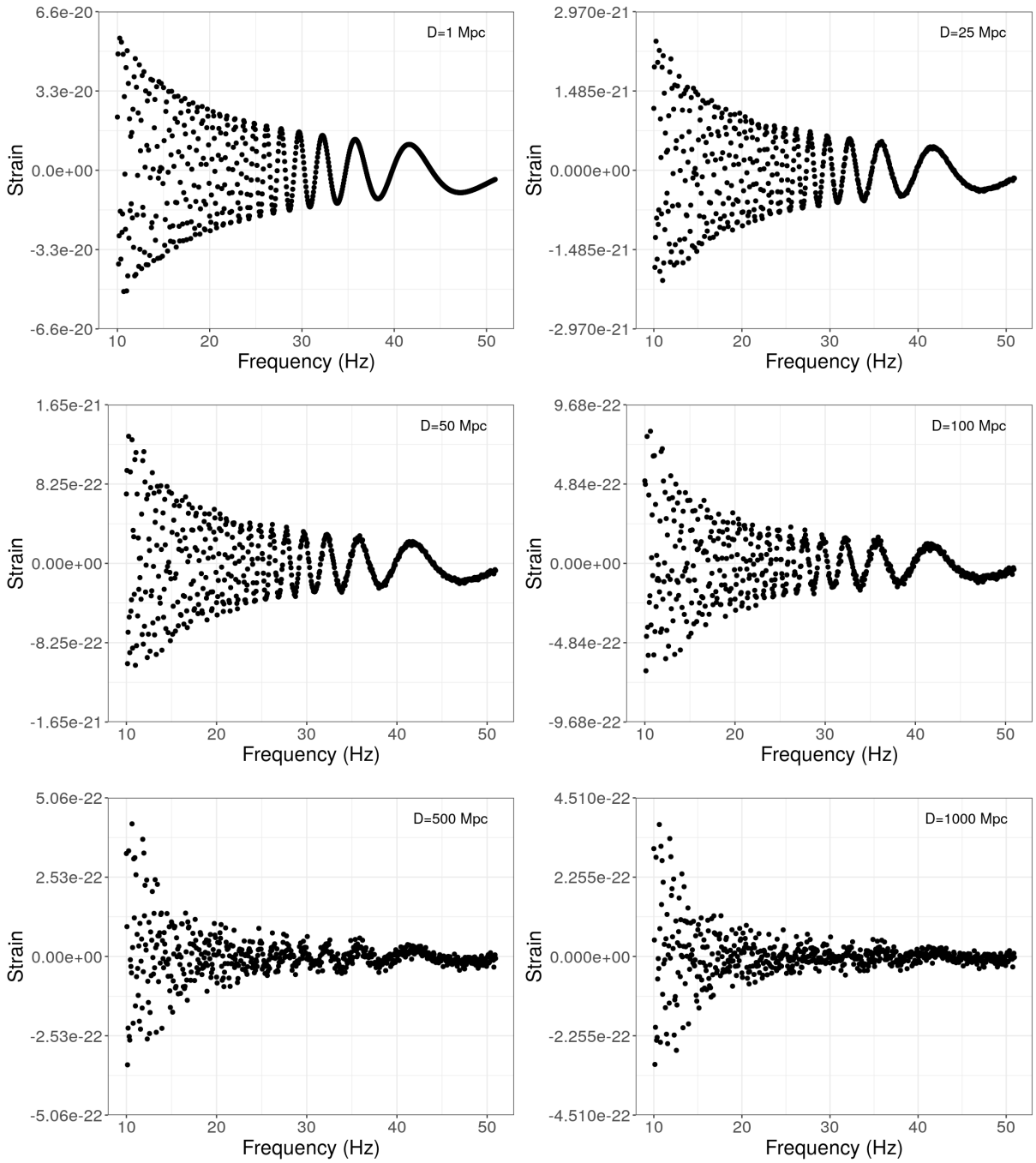

Figure 3.6: Visualization of six Frequency Domain Waveforms with $M_{chirp} = 17.17\ M_{\odot}$ at increasing distances (decresing signal-to-noise ratios), taken from our test dataset.

Data Pre-Processing

Before plunging into the actual machine learning inference, to implement it successfully and as realistically as possible, there is a need to perform two additional pre-processing steps.

The first step consists of cropping all the zeros in each waveform since each strain vector will contain a sequence of zeros after ringdown (TD) and before the merging frequency (FD). For higher chirp mass systems, this sequence gets increasingly large. Which means that, if we keep the zeros in the dataset, instead of making inferences of the chirp mass based on the shape of the wave, the algorithms will be making inferences based on the number of zeros after merging. That would be an unfair way of getting

good results without actually making proper and realistic inferences.

The second step regards the size discrepancy between each of the generated waveforms. Since the time it takes from the inspiral phase to the actual merging depends on the mass difference between the two GW sources, different chirp waveforms will span different time ranges. Consequently, this implies that the waveforms in our training and test sets will have unequal lengths. Since the ML algorithms only work with equal length instances, we need to perform a resizing of the whole dataset so that each wave has the same length.

In the **Time Domain** scenario, we choose to resize the waves by cropping the inspiral phase at 1 second before merging and cropping the ringdown phase of all waveforms to the size of the smallest waveform ringdown vector. This way, each time-domain waveform strain vector will have 4144 data points. 4096 data points before, and 48 data points after merging.

In the **Frequency Domain** scenario, we choose to resize the waves by cropping all the strain values for frequencies above $51\ Hz$, which corresponds to the limiting frequency domain of the shortest wave. This way, each frequency domain waveform will have 656 data points.

3.3.2 PCR for GW Parameter inference

One we get all our waveforms into their respective matrices we will have a training matrix composed of 256 instances and a test matrix with 255 instances, each corresponding to a matrix row. In TD we will have 4144 columns while in FD the matrices will have 656 columns.

Similarly to the cosmological density fields case, the next step consists of performing dimensionality reduction. This step works in the same way as in the density fields case, we use the same R function described previously with scaling and centring. After the projection, we will have a compact representation, where each waveform is represented by 256 PCs, regarding the training set.

As you may expect, since in this case, we are dealing with much shorter matrices, than in the N-body case, the PCA projection is quite less computationally resourceful, being much faster. On the other side, since we have a much higher number of instances (e.g 256 instead of 23), we will have a final projection with a much higher number of Principal Components.

Having our compressed data representation, we can proceed to the actual machine learning regressions. And here is where this part of the work highly diverges from the N-body emulation scenario. That is because, in this context, we are not attempting to emulate waveforms, we are instead attempting to infer parameters concerning those waveforms given their strain vectors.

Consequently, instead of using the PCs as dependent variables, we will be using them as independent variables, and the chirp masses will be the dependent variables for prediction. So instead of predicting the PCs given a chirp mass value, we will be predicting the chirp mass value given the PC values. That implies a large shortcut in our pipeline since we are not forced to build a regression model for every single PC.

Given this fact, in the training process, we build a single regression model for each of the machine learning algorithms, providing the full PC matrix containing the independent variables and the full chirp mass vector containing the dependent variables.

Regarding the algorithms themselves, in this part of the work, we excluded the neural networks. Since in this scenario we have 256 dimensions/independent variables, as opposed to the N-body context where we have a maximum of 2 independent variables (e.g Ω_m and z), here the single-layer neural network highly under-performs. To apply neural networks to this case with some insurance of success, a multi-layer deep neural network must be implemented, which is outside of the scope of this thesis. Recall that

in the N-body context, just by adding 1 independent dimension (z), the neural network changes from the best performing algorithm to one of the worst-performing ones.

Once trained and optimized, we store the models in a list to be given afterwards to the `predict` function together with the set of PCs representing the new test waveforms. To that, we need to first project the test waveforms in the PC basis of the training set. We achieve by applying the centre and scale vectors provided as outputs of the `prcomp` coupled with the PCA rotation matrix to the test waveforms, obtaining the PC vector for a specific waveform as:

$$\boldsymbol{PC_{new}} = \boldsymbol{L^T}\left(\frac{\boldsymbol{t_{wf}} - \boldsymbol{c}}{\boldsymbol{s}}\right) \tag{3.3.1}$$

where, as in the Density fields case, $\boldsymbol{L^T}$ corresponds to the loadings (or rotation) matrix provided by the PCA, $\boldsymbol{s}$ the PCA scaling vector, $\boldsymbol{c}$ the PCA centering vector and $\boldsymbol{t_{WF}}$ to the test waveform vector for chirp mass inference.

After having our 255 inferred chirp masses is time to proceed to the evaluation of the algorithms, which will be explained, together with the optimization process in the next Subsection.

3.3.3 Optimization and Performance

Similarly to the N-body case, here we also perform the so-called *internal* and *external* optimization process. However, in this case, we add an additional optimization process, regarding the number of PCs we want to include in the pipeline. In other words, here we also perform a Principal Component optimization.

PC Optimization Process

Since we have an extremely large amount of PCs in this case, we expect that most of them contain redundant or irrelevant information. Given that assumption, it should be possible to find an ideal set of PCs containing the most relevant pieces of information to be processed by the ML algorithms, making their regressions far more accurate.

To find that optimal amount of PCs, we build a loop spanning the whole PC range (e.g 256 iterations). In each iteration, we perform the regressions and evaluate the algorithms inferences, storing them in a vector, while using an increasing number of PCs cumulatively. After having that vector, we choose the optimal amount of PCs to be the one which provides the smallest error in the inferences.

Keep in mind that in this case, the optimization focuses only on the number of PCs used for training and testing. The hyper-parameters of the algorithms are set to their default values.

General Optimization

The internal optimization works in the same way as in the N-body case, the difference being that we only need to perform a single regression which already includes the total number of optimal PCs. Consequently, we get a single optimal value for each hyper-parameter instead of a multitude of optimal hyper-parameters, one for each PC regression loop. An additional difference, is in the chosen hyper-parameter ranges for the tree models, more specifically for the `mtry` parameter. Since in this case, we are dealing with data spanning a higher amount of dimensions, there is a need to translate that amount in the `mtry` parameter.

It is the external optimization process that differs from the N-body context. Since we are dealing with parameter inference, the power spectrum is not a possible choice for our evaluation metric. Instead,

following the lines of the internal optimization metric, we use RMSE for the external metric.

The remaining optimization process follows the same lines as in the N-body case. For each optimal range found in the internal optimization, we evaluate it using different schemes (CV-8, CV-10, CV-12, CV-23 and 20-repeat bootstrap).

Tree Models Optimization Specifics

Regarding the specifics of each algorithm, for SVM the process works exactly like in the N-body scenario. It is the optimization process of the tree models which slightly deviates from the previous case.

Compared with the N-body scenario, the layout of the optimization process is the same. We first perform a grid-search over the number of trees, fixing the optimal value. Afterwards we proceed to optimize the remaining two relevant parameters, `mtry` and `nodesize` for Random Forest, `mtry` and `numRandomCuts` for Extremely Randomized Trees.

As already mentioned, the main difference in the GW parameter inference scenario resides in the high number of features/dimensions we are dealing with. Consequently, for this case, the parameter `mtry` becomes highly relevant for optimization.

Moreover, given that we first perform a PC optimization, depending on the optimal number of PCs, the range over which we test the `mtry` parameter will change. (E.g. For 20 optimal PCs, we should search `mtry` on a 0 to 20 range. For 40 optimal PCs, we should search it on a 0 to 40 range.)

If you recall that in the N-body emulation scenario we only needed to search two `mtry` values at maximum (`mtry=[1,2]`), it becomes clear that in this scenario the internal optimization process will, in principle, require a considerably higher amount of time and computation resources. Given that fact we decided to shorten the range over which we search for the parameters `nodesize` and `numRandomCuts`. Instead of searching over a [1;15] range, we will be searching on a [1;10] range.

With the specifics explained, the general process works in the following way.

After performing the grid-search over the number of trees, we switch to 10-fold cross-validation applying three additional searches over the parameters `mtry` and `nodesize/numRandomCuts`. First, we search the two hyper-parameters together, fixing the best-found values and the corresponding RMSE. Secondly, we set mtry to its default value and search only the parameters `nodesize` or `numRandomCuts` in the mentioned range. Thirdly, we fix `nodesize` or `numRandomCuts` to their default values and search only the parameter `mtry`. Finally, after storing the results of the three experiments, we chose the range which provides the lower RMSE and apply the remaining optimization schemes (8-fold CV, 12-fold CV, 23-fold CV and 20-repeat bootstrap) to it. After obtaining the results of the regressions, we define the best scheme as the one which provides the smallest error.

Chapter 4

Results

In this Chapter, we present the results of our work. We first show the results concerning the N-body dark density field estimations, and following that, the results regarding the GW parameter inference.

In both cases, we begin by showing the results of the Principal Component decomposition, followed by the results of the optimization process for each algorithm. Finally, we will present the results of the final regressions, comparing the performance of the algorithms considering a series of error metrics.

All the 2D plots in this Chapter are plotted using the R package `ggplot2` [65].

4.1 Cosmological Density Fields

In this Section, we present the results regarding the cosmological density fields. We divide our results in two scenarios, in the first we attempt to estimate the density fields given a single free parameter, the dark matter density (Ω_{dm}), while in the second we attempt to perform the emulations introducing an additional free parameter, the redshift (z).

4.1.1 Ω_{dm} Estimators Performance

We will begin by presenting the results concerning the one-parameter estimation scenario.

We start by presenting results regarding the PCA, focused on our $z = 0$ dataset. We first show how different PCs behave with respect to the Ω_m values. Then, we also present results on the importance of the variance explained by each PC, how the density field looks when reconstructed only with a limited amount of PCs and how the Power Spectrum behaves for different PCA reconstruction scenarios.

Following the PCA, we present the results of the optimization process. Here we compare the final regressions against the $\Omega_m = 0.309$ simulation output. This comparison is performed at four different redshifts ($z = (0, 0.5, 1, 10)$), using both the Power Spectrum and Bispectrum as metrics. We also show results for the MOD evaluation metric, stressing the differences from the correlation function results.

Finally, we present the results of the FPCA implementation, comparing the estimation performance of the method against the supervised learning algorithms.

Principal Component Analysis

In Figure 4.1 we show the behavior of 9 PCs with respect to Ω_m. It is important to consider that the PCs are discrete in the sense that we do not have a continuous distribution of PC scores; we have one PC score for each Ω_{dm} observation. Thus the interpolated lines are just a tool to help the visualization.

Although it is not trivial to interpret the physical meaning of each PC, it is possible to draw some conclusions by looking at the overall distribution. The most striking aspect is the fact that for higher PCs, the score distribution becomes sinusoidal with decreasing amplitude and increasing frequency. The decrease in the amplitude is indicative that higher PCs contain less information, which is expected since they should correspond to a lower proportion of variance in the data. The increase in the frequency

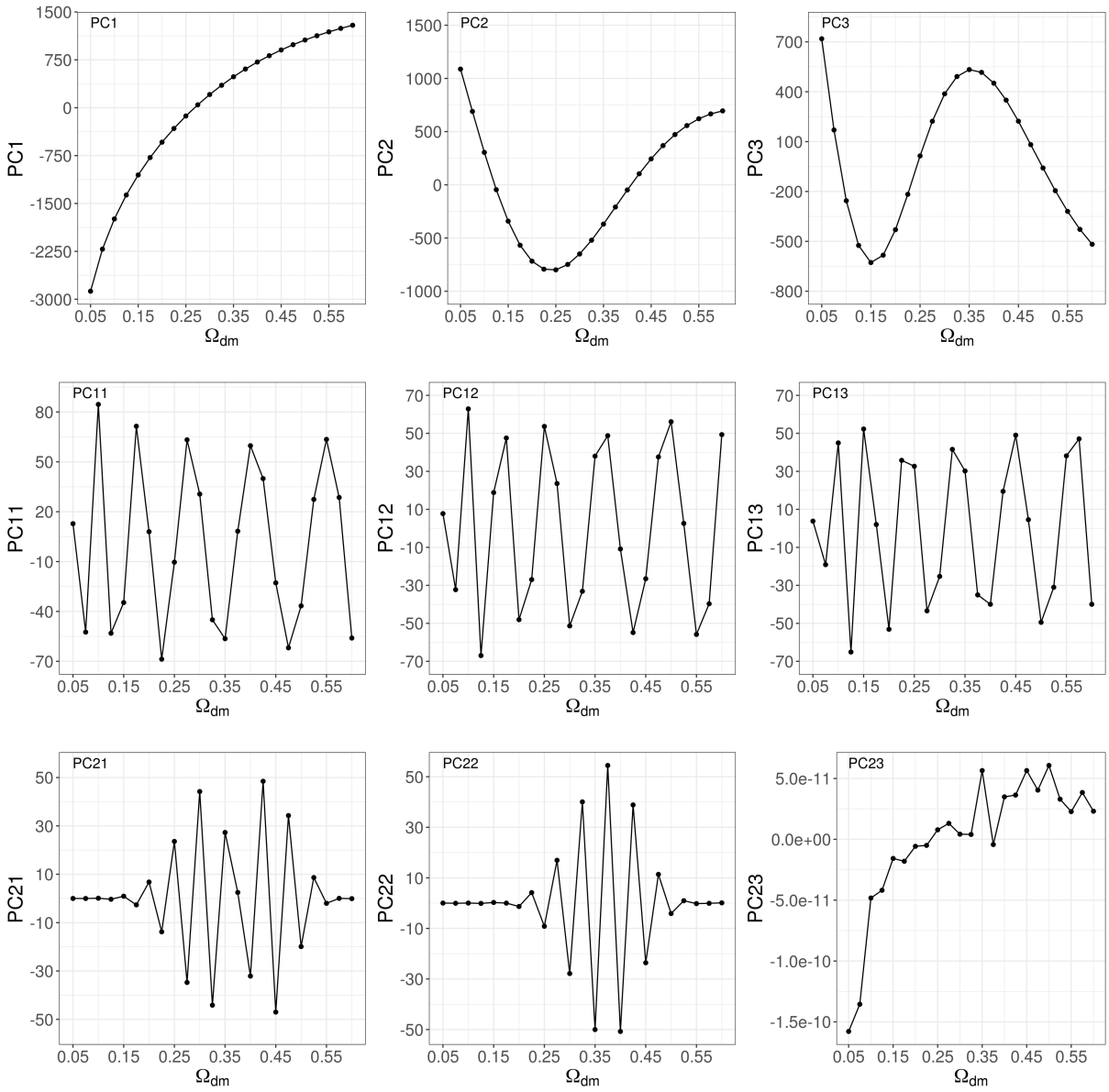

Figure 4.1: Plots showing the behaviour of nine increasing Principal Components with respect to Ω_{dm}. The PCA projection was performed on the single free-parameter (Ω_{dm}) dataset.

could suggest that higher PCs contain less physical information, describing properties which are less related with the dark matter density and more related with variance corrections to the initial PCs, and thus making their values highly oscillatory for different Ω_m's.

We can see further evidence that higher PCs contain less relevant information of the data by looking at Figure 4.2, showing the proportion of the variance explained by each PC.

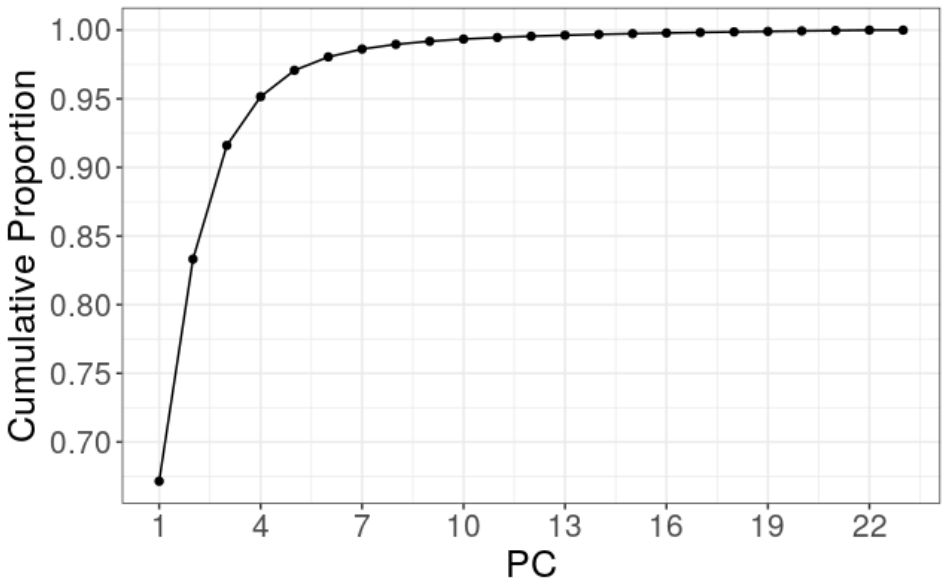

Figure 4.2: Cumulative Proportion of the Variance Explained by the Principal Components obtained through the projection of the single free-parameter (Ω_{dm}) dataset.

We can see that more than 95% of the variance in the data is contained just in the first 4 PCs, the first one explaining around 67%. By the time we add the 10th PC around 99% of the variance is explained, and the remaining PCs summation seems to produce almost no changes in the variance.

We also show in Figure 4.3, as an example, two sets of three images representing the PC reconstruction of the $\Omega_m = 0.3$ density field at $z = 0$. In the upper images we show the cumulative reconstruction of the density field for 1, 12 and 23 PCs while in the bottom images we show the single PC reconstruction, for PC1, PC12 and PC23. These 3D plots were obtained using the same tool as in Figure 3.3, which receives as the main inputs the coordinates x,y,z and the density array, which in these Figures is log-scaled.

These Figures show that in the cumulative summation scenario it is clear that adding PCs is equivalent to adding density to the distribution. The filaments seem to remain unchanged for the three cubes, which could be due to their information being already contained in the first PC. It seems that the following PCs are mainly adding density in the void regions if we notice that the colour of the filaments themselves remain nearly unchanged through all of the reconstruction.

Regarding the single PC reconstruction, we can observe that there are hardly any changes in the images. That can be expected since each PC carries much less information than the overall sum, and thus we can expect less noticeable changes. However, in the case of PC23, the contrast between the filaments and the voids seems to be more noticeable, indicating that higher PCs might be adding more information on the voids.

Finally, we also show the effects of the PCA reconstruction in the Power Spectrum of the Density Cube in Figure 4.4.

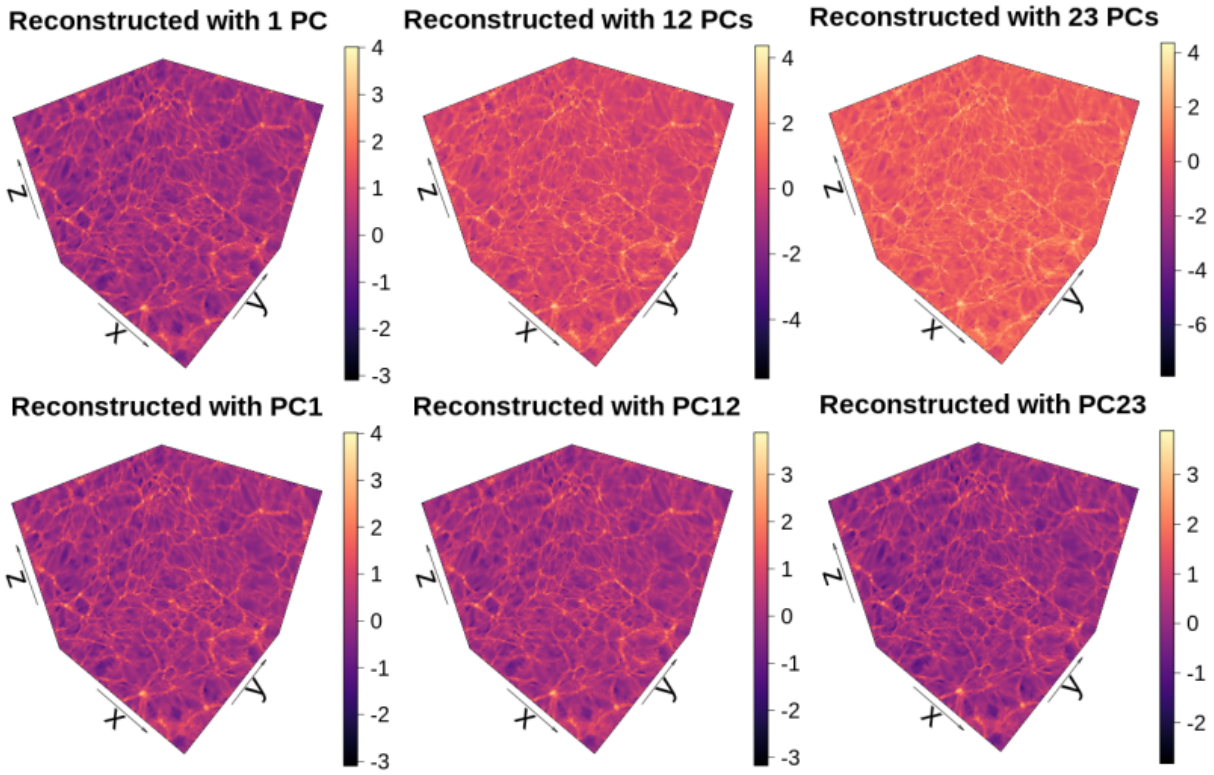

Figure 4.3: 3D visualization of the Reconstructed Cubes using increasing PCs, for $\Omega_m = 0.30$ at $z = 0$. **Top**: Cumulative PC Reconstruction. **Bottom**: Single PC Reconstruction.

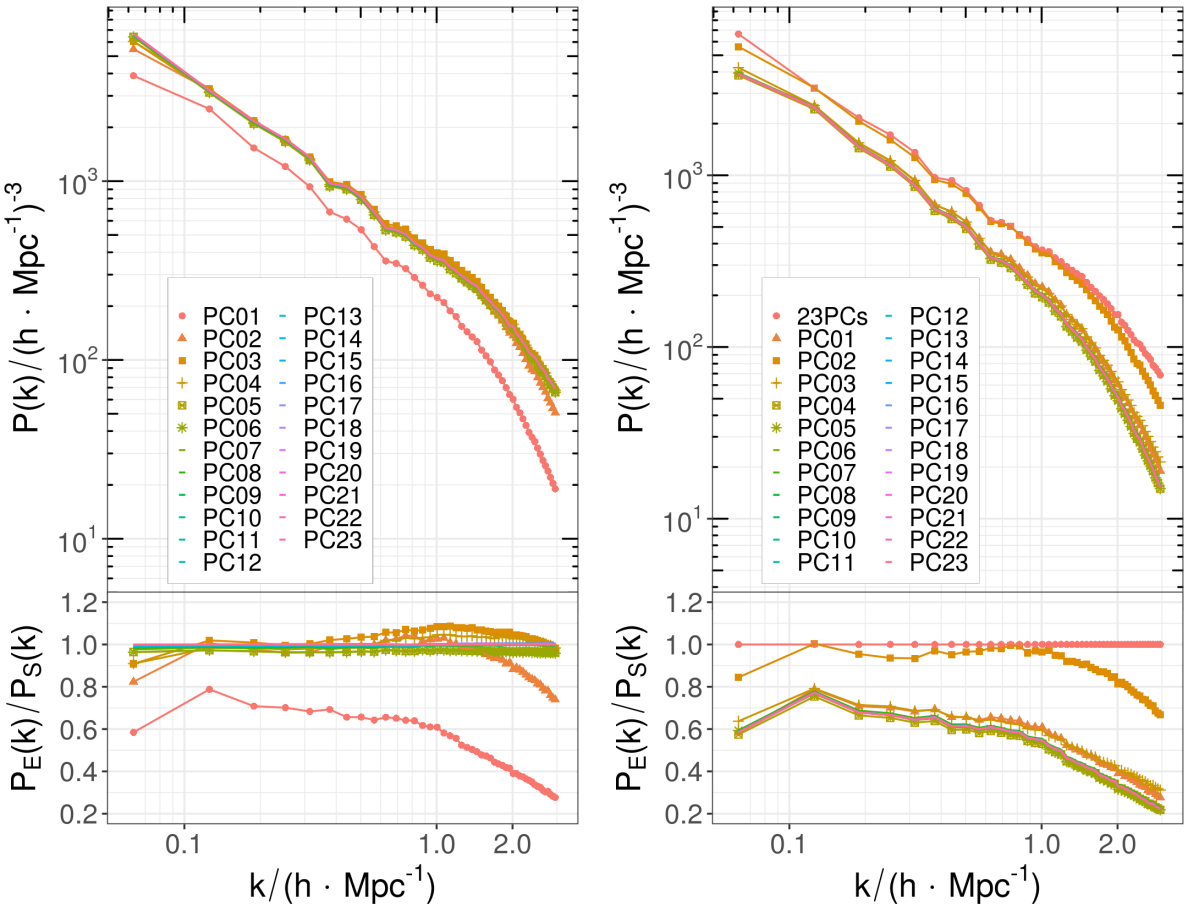

Figure 4.4: Plots showing the Power Spectra of the reconstructed density field for $\Omega_{dm} = 0.3$ and $z = 0$. **Left:** Cumulative PC reconstruction. **Right:** Single PC Reconstruction. **Top Panel:** Power Spectrum Curves. **Bottom Panel**: Power Spectrum Ratio ($P_E(k)$ = Estimated;$P_S(k)$ = Simulated).

In the left, we can see the case of the cumulative PC reconstruction while at the right we see single the PC reconstructions. The top panels correspond to the Power Spectrum curves while the bottom ones correspond to the ratio between the power of the reconstructions and the full 23 PC cube. The full cube is represented by the labels $PC23$ and $23PCs$ at the left and right plots, respectively.

For the left plot, in the top panel, we see an almost full superposition of the curves at around $PC \geqslant 3$ while in the bottom one we can still see some divergence at the 6th PC reconstruction and by the time we reach the blue and pink segments, the ratio is practically one. This is expect if we recall our previous discussion on the variance explained. One interesting effect that we can notice in both panels is the drastic difference in the addition of the second PC, which seems to be a crucial component in the explanation of the original, uncompressed data.

The right plot, with the power spectra of the reconstruction using a single PC, seems to be an almost inverted version of the cumulative reconstruction. This can be understood considering that since as we proceed to the highest PCs, instead of having an almost complete density field like in the cumulative PC reconstruction, we have density fields lacking the most relevant components. And thus it is expected that the curves get farther away from the full reconstruction. The most unexpected feature in these results is the change in roles of the PC1 and PC2. Here we can see that a density field reconstructed with just the second PC is closer to the full reconstruction than the density field reconstructed with only the first PC. This result is consistent with the left plot, since we see the PC2 bridging the gap to the full reconstruction with high dominance over the remaining PCs, but on the other side, we also see PC1 bridging the initial 60% difference gap. This seems to show that a more accurate power spectrum reproduction does not necessarily imply an higher proportion of the variance explained.

Optimization Results

Since we are applying our pipeline in four different redshifts, meaning that we are training our algorithms with simulations using the same set of Ω_m's at distinct redshift snapshots, we employ independent hyper-parameter optimizations at each redshift.

We present the results in four tables, one for each algorithm, where we show the results for the best optimization schemes at each redshift and for each hyper-parameter.

For the tree-based models, ET and RF, there are two distinct optimization steps. In the first step we perform a grid-search over the number of trees; this step fixes the optimal value for the number of trees. In the second step we apply the chosen optimization schemes over the remaining hyper-parameters. Often, the first step is omitted in the tables presented in this Subsection, except for the cases in which the best model corresponds to a simple grid-search over the number of trees.

We can see the results for Random Forest in Table 4.1.1. For $z = 0$ the best result was the one where we set the trees to `ntree=1000` and `nodesize` to its default value `nodesize=5`. For $z = 0.5$ and $z = 1$, we first performed the regular grid-search fixing `ntree=1000` as the best performing case. Afterwards, we proceeded to apply the remaining optimization schemes to chosen `nodesize` range ([1;15]) with the best performing scheme being the 23-fold CV, usually called *Leave-one-out Cross-Validation* (LOOCV). Finally, for $z = 10$, our initial grid-search found `ntree=2000` as the best performing model. In the second step, we managed to further improve the model finding that 20-repeat Bootstrap over-performed Cross-Validation.

We show the results for the Extremely Randomized Trees in Table 4.1.2. Interestingly, the results were the same as in RF regarding the first step of optimization. For the second step, they were quite different. Here, the only case where the best results corresponded to a Grid-Search was for $z = 10$, where

Table 4.1.1: Optimization results for the best found Random Forest 1D regressions, at different redshifts.

RF	**z=0**	**z=0.5**	**z=1**	**z=10**
ntree	1000	1000	1000	2000
nodesize	Default	[1;15]	[1;15]	[1;15]
Method	Grid-Search	LOOCV	LOOCV	20-rep Bootstrap
PS_r	4.04	2.95	4.01	0.44

the best model was found for `ntree=2000` and `nodesize` set to its default value. For the remaining redshifts, the second optimization step managed to always improve the models compared to the first, where the best performing schemes were *LOOCV* for $z = 0$ and *8-CV*, for $z = 0.5$ and $z = 1$, respectively. Regarding the neural networks, we show the results in Table 4.1.3. Here the results are quite monotonous,

Table 4.1.2: Optimization results for the best found Extremely Randomized Trees 1D regressions, at different redshifts.

ET	**z=0**	**z=0.5**	**z=1**	**z=10**
ntree	1000	1000	1000	2000
numRandomCuts	[1;15]	[1;15]	[1;15]	Default
Method	LOOCV	8-CV	8-CV	Grid-Search
PS_r	13.40	3.84	10.19	0.56

the best-performing scheme being always 10-CV with `decay` set to its default value (`decay=0`). The only differences in performance concern the search range for the hyper-parameter `size`. We found the optimal range to be [1;75] for $z = 0$ and $z = 1$, and [1;100] for $z = 0.5$ and $z = 10$. Finally, we show the results for

Table 4.1.3: Optimization results for the best found Neural Network 1D regressions, at different redshifts.

NNET	**z=0**	**z=0.5**	**z=1**	**z=10**
size	[1;75]	[1;100]	[1;75]	[1;100]
decay	Default	Default	Default	Default
Method	10-CV	10-CV	10-CV	10-CV
PS_r	0.57	0.59	0.75	0.44

the support vector machines in Table 4.1.4. As mentioned in the previous Chapter, the best kernel was always found to be the radial basis one. And for this case of one-parameter estimation, the best range of search found was always R_2. For $z = 0$ and $z = 0.5$, the best found optimization scheme was 8-CV, for $z = 1$ we found 10-CV with gamma set to its default value ($gamma = 1$), and for $z = 10$ the best scheme was a 20-repeat Bootstrap with *cost* set to its default value $cost = 1$.

Table 4.1.4: Optimization results for the best found Support Vector Machine regressions, at different redshifts.

SVM	**z=0**	**z=0.5**	**z=1**	**z=10**
Kernel	Radial	Radial	Radial	Radial
cost	R2	R2	R2	Default
gamma	R2	R2	Default	R2
Method	8-CV	8-CV	10-CV	20-repeat Bootstrap
PS_r	0.99	5.61	2.61	1.13

Power Spectrum and Bispectrum Evaluation

In this Section, we present our main results. We show how each of our estimated density fields compare against the simulated density field using their power spectra and bispectra as comparison metrics. We present the results of the optimal regression models for four redshift regressions.

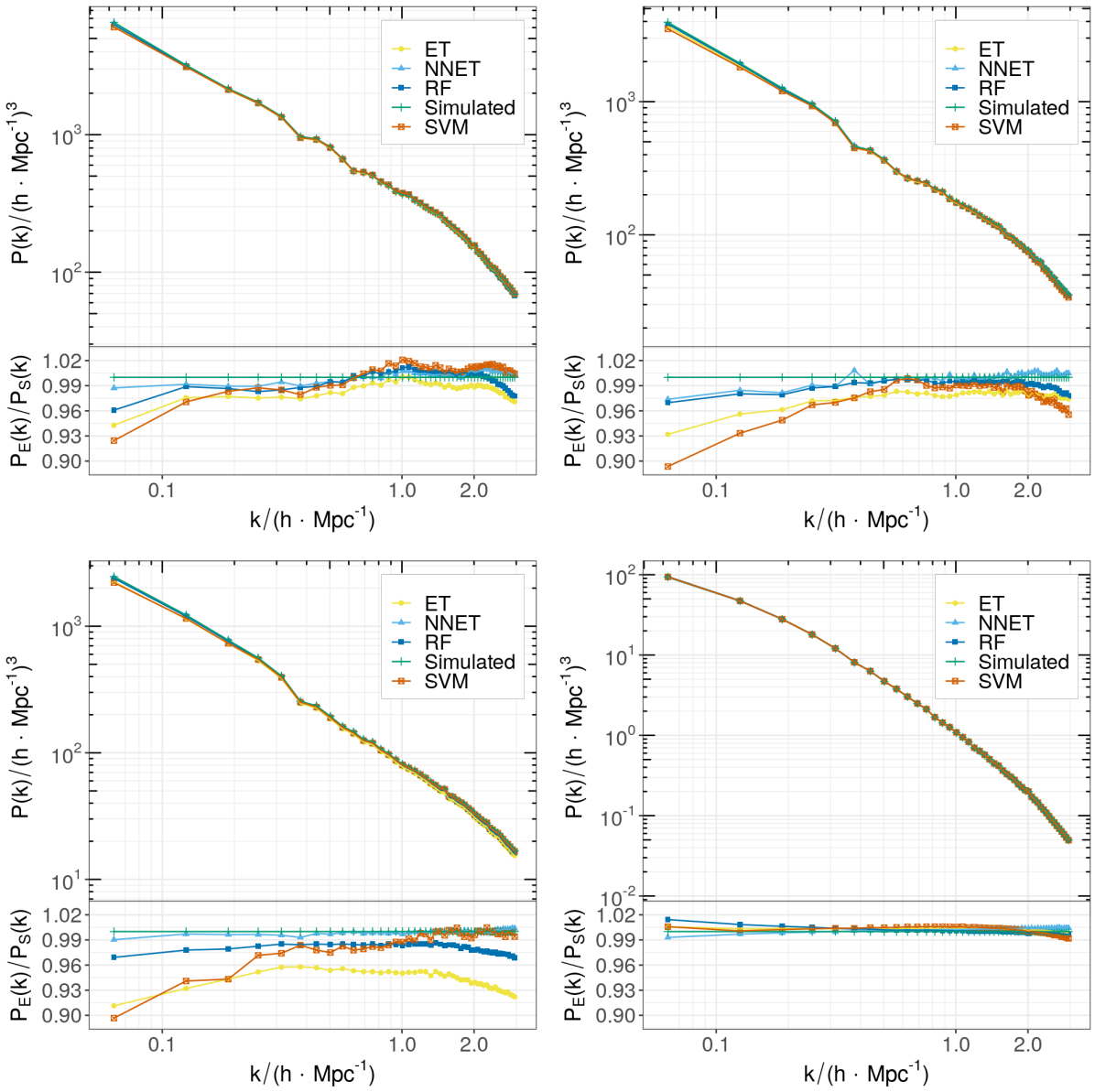

Figure 4.5: The Power Spectra of the estimated Density Fields, at each redshifts and using the four algorithms. From top left to bottom right we show the results concerning the implementation of our pipeline for simulations at z=0, z=0.5, z=1 and z=10, respectively. **Top Panels:** Power Spectrum Curves. **Bottom Panels**: Power Spectrum Ratio ($P_E(k)$ = Estimated;$P_S(k)$ = Simulated).

The results are shown in Figure 4.5. The same analysis shown in Figure 4.4 applies to these results. On the top panels, we represent Power Spectra of each estimated density field and also of the simulated field. On the bottom panels we represent the ratios between the estimated and simulated powers. We show overlapping results of the Power Spectrum for four estimations: Extremely Randomized Trees (ET), Neural Networks (NNET), Random Forest (RF) and Support Vector Machine (SVM).

Concerning the top panels, there seems to exist an almost complete overlap of all powers considering $k > 0.1$ hMpc^{-1}, while at the lower k region, we begin to see the estimated power diverging from the

simulated one. There is an exception, which corresponds to the $z = 1$ case, where we can see that at least the ET estimation seems to deviate from the simulation curve for most of the domain.

Overall, the results of our emulator seem to follow reasonably well the results of the simulation. We will now analyse more carfully the bottom panel. We will first make remarks about the overall performance at different redshifts, followed by a comparison of the algorithms, through the analysis of their powers in the left and right extremes of the k domain, as well as in the central regions.

Regarding the redshift, we should expect the performance of the algorithms to increase with it. That is because, as we travel in cosmological time, the structure formation process gets increasingly non-linear. Consequently, since looking at a field at higher redshifts corresponds to looking at it at past times, we are thus looking at a cosmological time frame where the structure formation dynamics were more linear than in the present, which translates in a field where it's density cells are more linearly correlated. That should imply an easier compression process, and consequently a simpler learning process.

However, we don't see that principle holding perfectly in our results. If we consider the SVM and the tree models, their performances seem to worsen from $z = 0$ to $z = 1$, albeit it improves significantly at $z = 10$. That is particularly striking for the ET and RF estimations at $z = 1$, which are far worse in all of the k domain compared to the other redshifts. The exception to this unexpected rule is the NNET estimation, that shows a consistent improvement in performance from $z = 0$ to $z = 0.5$ (even a slight improvement around $k = 1$ hMpc^{-1}) and a clear improvement in all of the domain from $z = 0.5$ to $z = 1$ and especially at $z = 10$.

Now, before comparing the performances of the algorithms, we will give an additional remark regarding one feature that we can see at all redshifts and for all estimators. We can observe that for all cases, the algorithms seem to have worse performances at larger scales ($k \leq 0.6$ hMpc^{-1}), which can be as bad as a 10% difference in $k \leq 0.1$ hMpc^{-1} for algorithms as the SVM, at $z = 0.5$ and $z = 1$. That is understandable since larger scales are not representative of the overall training dataset. We can further support that argument by pointing out the scarcity of data points for the methods to learn about such scales. The algorithms also tend to have worse performances at the other extreme of the k domain ($k \geq 2$ hMpc^{-1}. A hypothesis to explain this behavior is that as we increase the k value, we are considering increasingly smaller scales where highly non-linear dynamics occur due to advanced stages of structure collapse.

Now, let's focus on the algorithms and their differences. Beginning with the tree models, we can notice that their power spectra ratio curves have very similar shapes. That is expected since the extremely randomized trees can be seen as a particular case of a random forest and it thus works in the same way both for learning and predicting, as we explained in Chapter 2. Nevertheless, the RF estimator is consistently better in all of the domain and at all redshifts, except at $z = 10$ and $k \geq 0.3$ hMpc^{-1}.

Regarding SVM, this algorithm highly under-performs when we consider the large scale extreme of the domain ($k \leq 0.5$ hMpc^{-1}). For the rest of the domain, this algorithm works quite well, achieving a difference of less than 1% for $k \geq 1$ hMpc^{-1} in all cases, over-performing both tree models in this region of the k domain.

Finally, the best performing algorithm in this case are the neural networks. We observe differences within 1% for $k > 0.3$ hMpc^{-1} at all redshifts. The worst results are, once again, at the large scale extreme and from $z = 0$ to $z = 0.5$. This is impressive, because these results show that it is possible to attain very low errors even at extreme scales as $k \geq 2$ hMpc^{-1} and nearing $k \sim 0.3$ hMpc^{-1}, while using one of the most simple types of neural networks, a single-layer network. We note, however, that this is only possible because of the compression step that was adopted here.

To conclude this part of the analysis, these results show that overall the methods managed to achieve high accuracies. We saw that the performance of the estimations tends to worsen at the extremes of

the k domain, especially at the large scale extreme (low k). We also noticed an unexpected decrease in performance for the tree models at $z = 1$. These unexpected decrease in performance should be better analysed, since it could due to some statistical behavior due to a possible non ideal local minimum obtained in the process (prone to improvement with a more sophisticated parameter optimization process), or it could be of physical nature, concerning the types of structures that may appear in the training set at such redshifts. This remains to be studied in a future work.

Lastly and in order to gauge the usefulness of our optimization process we also show the results for the same estimations, but with all the algorithms set to their default values, with no prior optimization. These results are shown in A.1 of the Appendix. By comparing these results from the Appendix with the optimized results, the improvement is clear. It is however important to keep in mind that NNET does not have any default parameter for its size, and thus it was defined to one. This means that this non-optimized NNET is a single perceptron regressor, and thus this is a reasonably unfair comparison to other methods and to the optimized version.

In Figure 4.6 we present the results of the Bispectrum analysis. Overall we can notice the same trends observed in the power spectrum. NNET tends to be the best performing algorithm and the under-performance of ET is also still present.

We see that at the extremes of the θ domain, the algorithms also tend to under-perform. This result being particularly striking for the extreme of large θ. This is quite an unexpected result, due to an inconsistency with the results in Figure 4.5. Since in the triangular configurations used in order to calculate the Bispectrum, we have a fixed k_1 and k_2, this should imply that increasing θ would be equivalent to increasing k_3. Consequently, large θ's should correspond to large $k's$, implying small scales.

Thus, the results seem to be inverted from the power spectrum to the bispectrum picture, in terms of the relation between the algorithms performance and the scale of the density field. In the power spectrum analysis, we see that the algorthms tend to highly under-perform for large scales (small $k's$). On the other side, in the bispectrum analysis, we see that the algorithms tend to highly under-perform for small scales (large θ implies large k).

It is important to remember that the Bispectrum probes a different set of physical properties than the Power Spectrum. More specifically, the Bispectrum is more sensitive to deviations from Gaussianity. Since these deviations occur mainly at small scales, and are minimal at large scales, it is expected that we should have more difficulty in reproducing the bispectrum at the former scales. Another way to look at this problem is by looking at the actual Power Spectrum and Bispectrum curves. We see that the curves seem to be have opposite behaviors with respect to scale. In the Power Spectrum analysis results, we see more power at large scales, and less power at small scales. In the Bispectrum picture, we see the opposite. This seems to imply that the error in estimation at a certain scale has a certain degree of positive correlation with the magnitude of the power at that scale.

MOD Evaluation

In Figure 4.7, we show the results of comparing our estimated fields with the simulated one using the MOD metric defined in equation 3.2.5. This equation compares the simulated field to the cumulative PC reconstruction of the estimated ones. It is clear that these results differ quite dramatically from the Power Spectrum results. First, it is possible to observe a consistent improvement in the overall performances, with respect to the redshift. That is the exact behaviour that we should expect theoretically, as mentioned in the previous Subsection. From $z = 0$ to $z = 0.5$, we see a consistent improvement on the tree models and SVM estimations, while the neural networks which maintain roughly the same performance. From

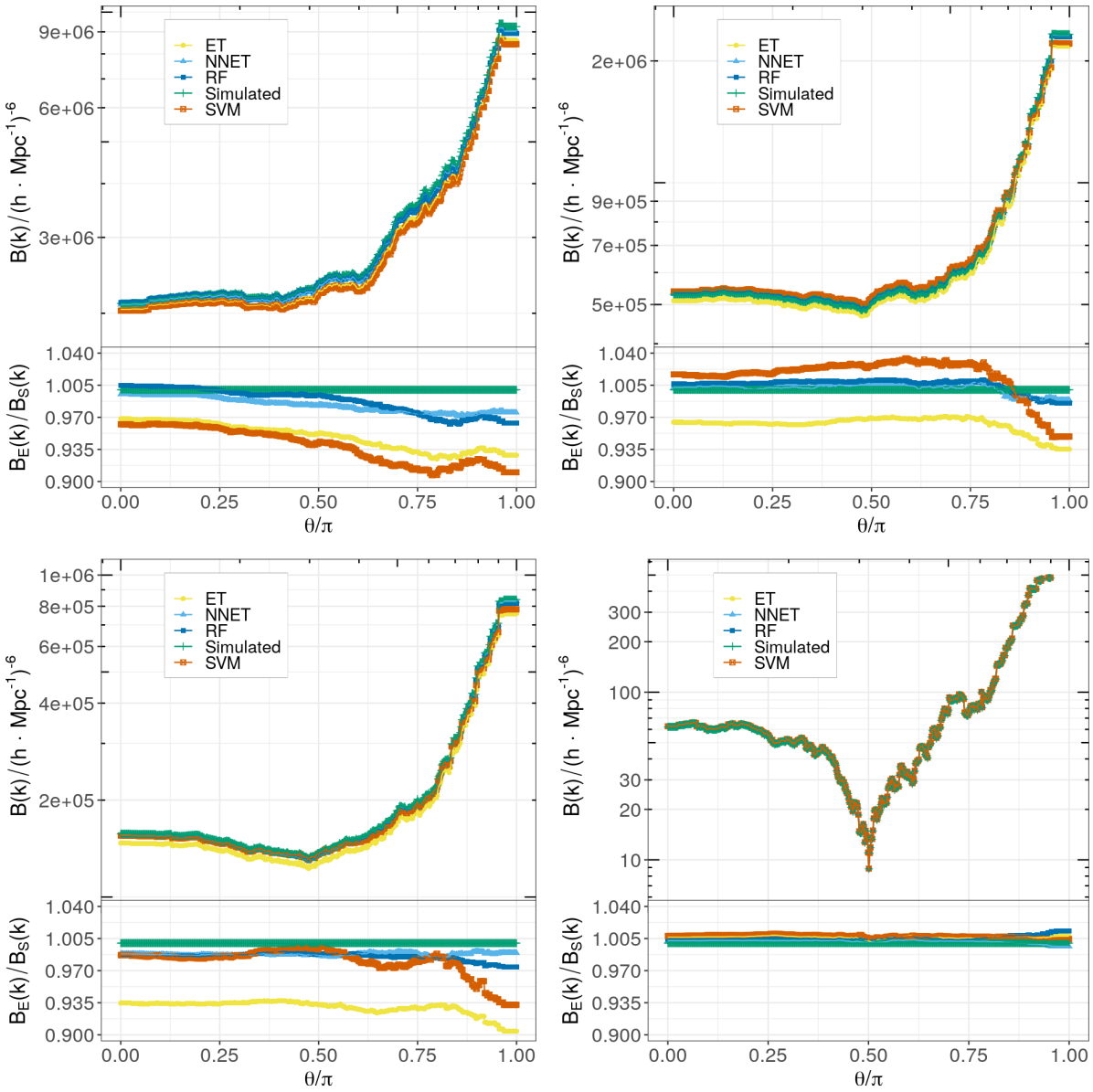

Figure 4.6: The Bispectra of the estimated Density Fields, at each redshifts and using the four algorithms. From top left to bottom right we show the results concerning the implementation of our pipeline for simulations at z=0, z=0.5, z=1 and z=10, respectively. **Top Panels:** Bipectrum Curves. **Bottom Panels**: Bispectrum Ratio ($B_E(k)$ = Estimated;$B_S(k)$ = Simulated).

$z = 0.5$ to $z = 10$, we see all the estimations improving, reaching the extremely low MOD values observed at $z = 10$.

Second, we see most of the algorithms changing roles in their performances. While in the power spectrum results, the neural networks are the best performing algorithm, here they seem to be the worst performing one. For all redshifts, we see an initial fast decrease in the error for the first 5 to 6 PC reconstructions, followed by an increase of the error as we add the higher PCs. On the other hand, the tree algorithms that have poor results when we consider the the power spectrum analysis, in the MOD analysis are the best performing one. This difference is particularly striking in the case of $z = 1$, where instead of observing the large decrease in performance of the tree models seen in the power spectrum picture, we see a consistent improvement.

The decrease in the error with increasing redshift is relatively easy to understand. It stems from two reasons. First, in MOD, we normalize the over-density differences by the number of density cells in the

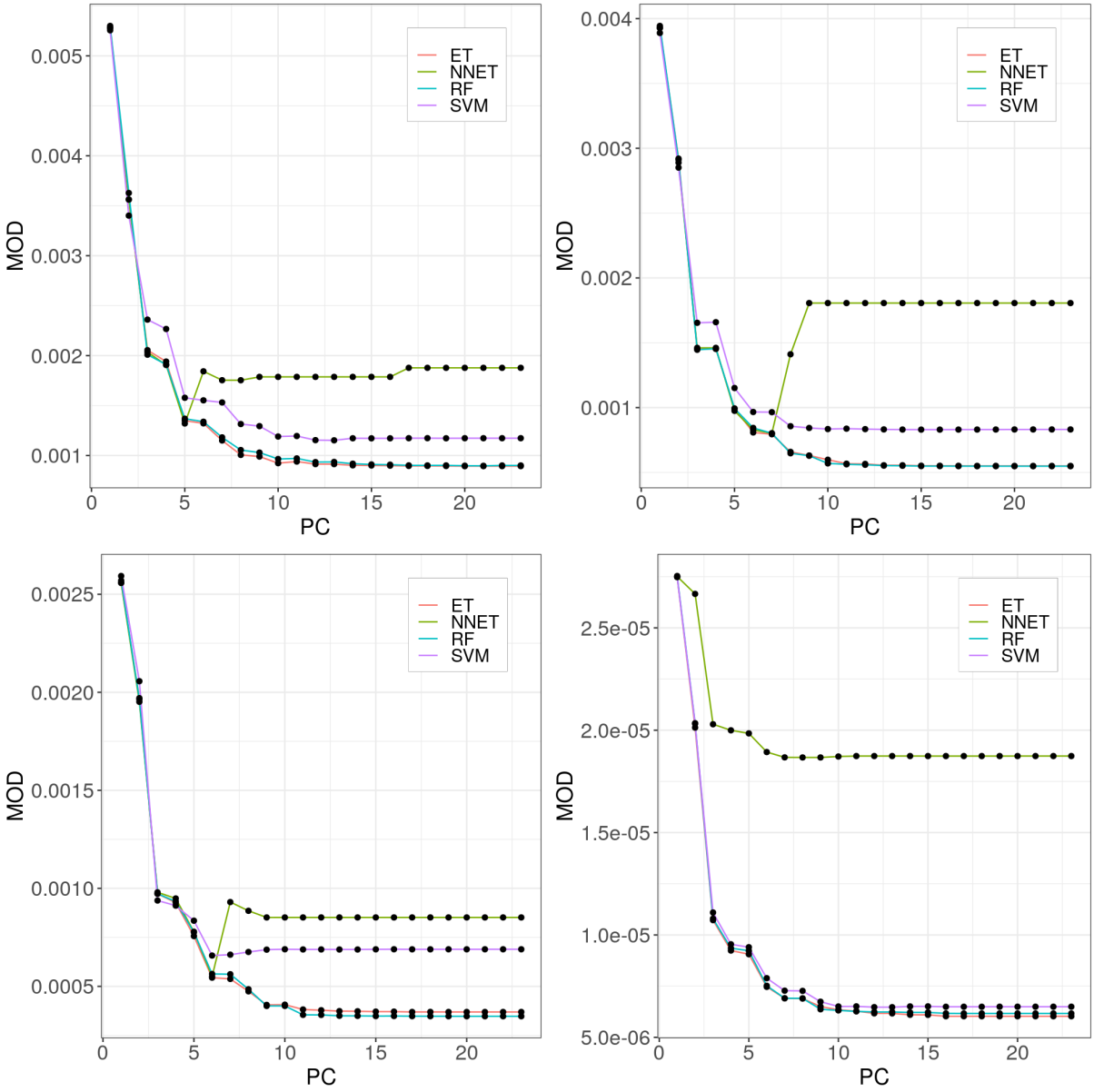

Figure 4.7: The Mean Over-Densities of the estimated Density Fields, at each redshifts and using the four algorithms. From top left to bottom right we show the results concerning the implementation of our pipeline for simulations at z=0, z=0.5, z=1 and z=10, respectively.

cube. Consequently, as we analyze cubes at higher redshifts (implying lower density), we will necessarily obtain lower MODs, due to the density cell normalization, even if the percentual estimation differences in each cell are large. Second, as we commented before, there is a relation between the increase in linearity at high redshifts, with the increase of the predictive capabilities of our estimators as we are employing a linear compression step with the adoption of a PCA.

The poor performance of the neural networks in these results, however, is less easy to understand. The possible hypothesis is that when looking at the power spectrum, we are looking at the statistical properties of the density field and not its absolute density values. Considering that, the neural networks could be performing estimations of structures with slightly different spatial distributions (equivalent to N-body simulations with distinct initial condition seeds but the same cosmology) while preserving the same statistical properties of the target density distribution (density correlations within a given scale).

The reason for the rise in the error with the addition of higher PCs could be explained by a combination

of the fact that we are dealing with a one-layer neural network with the fact that higher PCs seem to show less linear behaviour (as can be seen in Figure 4.1). One single layer is hardly enough to describe the non-linear behaviour of the higher components. Thus, to estimate these components with higher accuracy, a deeper network architecture should be implemented.

Finally, one feature that can be observed in general is the sharp drop in the error at the first 6 PCs, compared to the somewhat constant behaviour for the following ones. That is simply a consequence of the results in Figure 4.2, where we can see this first set of PCs explaining most of the variance.

FPCA Results

In Figure 4.8 we can see the results for the FPCA estimation, compared against the optimized NNET results. We can see that even when compared against the best performing algorithm of our previous

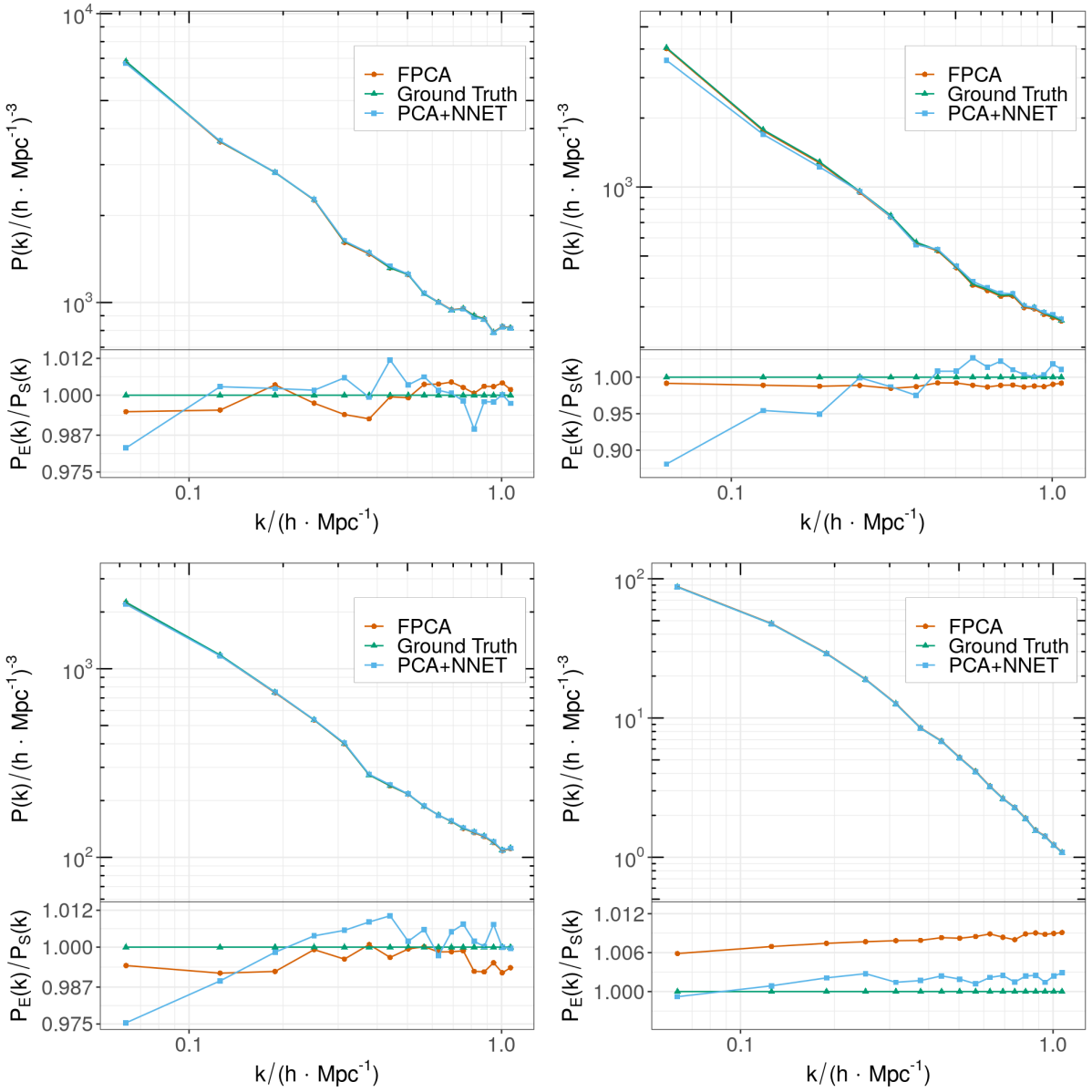

Figure 4.8: The Power Spectra of the estimated FPCA Density Fields, at each redshifts and compared against an optimized NNET. From top left to bottom right we show the results concerning the implementation of our pipeline for simulations at z=0, z=0.5, z=1 and z=10, respectively. **Top Panel:** Power Spectrum Curves. **Bottom Panel**: Power Spectrum Ratio ($P_E(k)$ = Estimated;$P_S(k)$ = Simulated).

results, FPCA seems to attain an even better performance. This holds for all z's except the $z = 10$ case, where FPCA is highly overestimating the power compared to NNET.

Additionally, FPCA seems to be more stable than NNET, in the sense that we do not observe the frequent oscillations in the power ratio that we can observe in the NNET case. That could be due to the smoothing process of the b-spline basis functions that takes place within the FPCA.

In addition to that, FPCA also seems to be an exception to the problem of estimation at large scales (lower k's). Although NNET seems to keep having decreased performances at the larger scale regime (except for $z = 10$, FPCA behaves quite well, maintaining higher consistency further up in the k domain.

Finally, one interesting thing to notice when looking at the error in terms of overestimation and underestimation of the power is the fact that NNET and FPCA seem to roughly exchange roles. For every z except $z = 10$, when one underestimates, the other overestimates. This seems to suggest that it would be possible to further improve the results by creating a model as an ensemble or a stacked generalization of these two methods [66]. Since, roughly, their average results seem to overlap with the simulated Power Spectrum.

The results for the Bispectrum are shown in Figure 4.9. For this case, the differences between both algorithm estimations are more nitid.

One one hand, both ratios seem to be more stable on the large scales. At $z = 0$ and $z = 1$, FPCA seems to be consistently better than NNET while the opposite happens for $z = 0.5$ and $z = 10$. The exception resides the smallest scales (large θ), where both algorithms seem to oscillate quite a bit in their bispectrum ratios.

These are quite remarkable results, showing that having the required computational power, one can shortcut the supervised machine learning application while obtaining possibly, even better results (as in $z = 0$ and $z = 1$ cases), at least for this size of simulations. Nevertheless, a more extensive analysis of the time taken by the FPCA pipeline, its estimation accuracy and how both scale with the size of the simulations would be required to take solid conclusions on the advantages of this new method, that has been scarcely adopted in astronomy and cosmology so far.

4.1.2 (Ω_{dm},z) Estimators Performance

Now we will present the results regarding the scenario of two-parameter density field estimation.

We present the results following the same logic as in the previous Subsection, but with some differences. We begin by showing the PCA compression results, focusing only on the proportion of the variance explained. This time, we present neither the relation free-parameter/PC nor the PC reconstruction translated in the Power Spectrum. The reason for the first omission is the additional dimension in our dataset, which would force us to plot a surface in both redshift and matter density space for each PC, which would translate in an excessive amount of information. The reason for the second omission is the large number of PCs we have in this dataset, which once again translates in an excessive amount of information contained in a plot with 92 overlapping power spectra curves. Following the PCA, we show the optimization results and the final regressions, again using the Power Spectrum and the Bispectrum, with final remarks on the MOD.

The main objective of this present Section is to demonstrate the validity of the proposed methods in a higher-dimensional context.

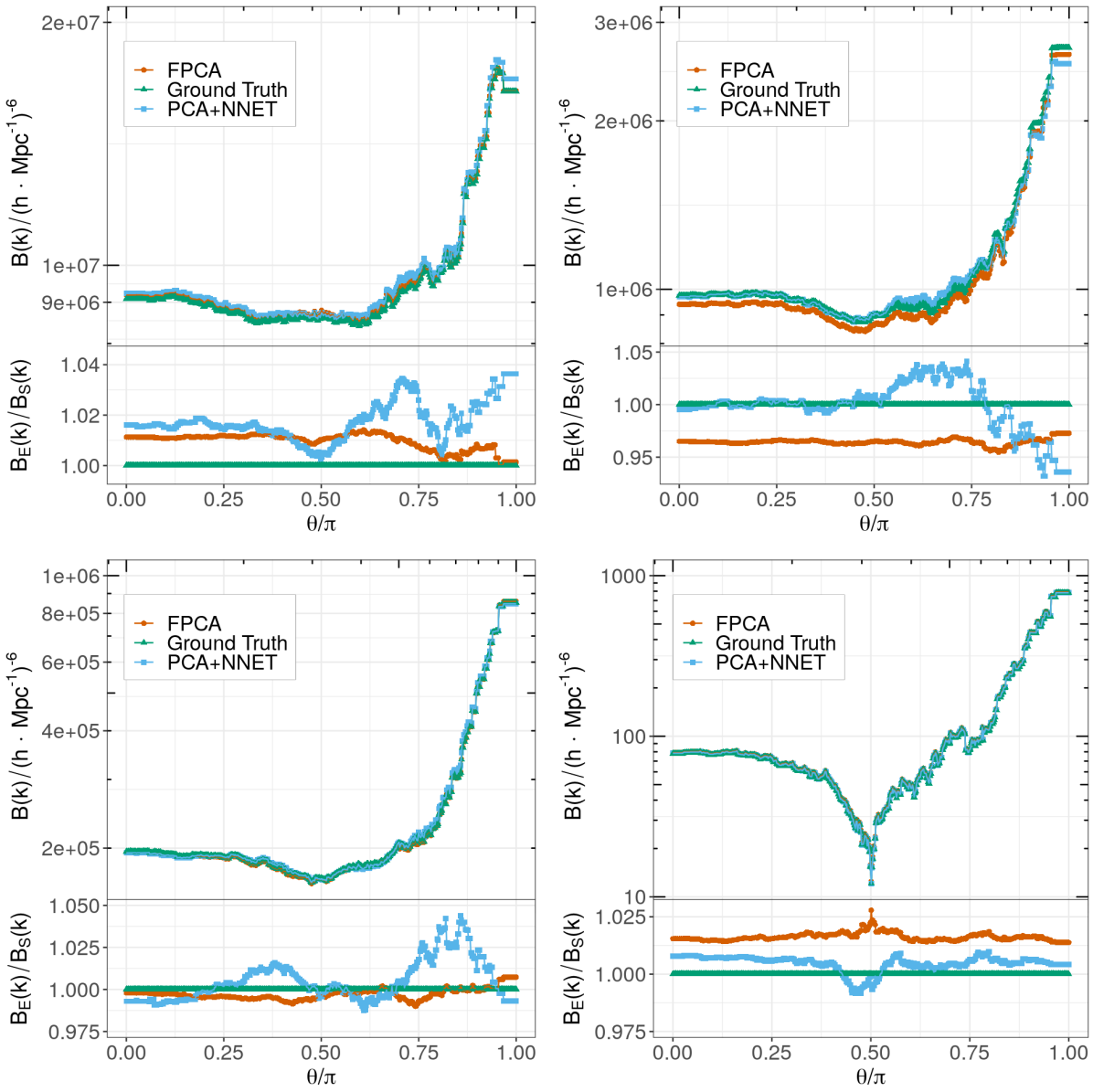

Figure 4.9: The Bispectra of the estimated FPCA Density Fields, at each redshifts and compared against an optimized NNET. From top left to bottom right we show the results concerning the implementation of our pipeline for simulations at z=0, z=0.5, z=1 and z=10, respectively. **Top Panel:** Bispectrum Curves. **Bottom Panel**: Bispectrum Ratio ($B_E(k)$ = Estimated;$B_S(k)$ = Simulated).

Principal Component Analysis

In Figure 4.10 we present the cumulative proportion of the variance explained by each PC, in our (Ω_{dm},z) dataset decomposition. This Figure shows that there is a considerably larger number of PCs than in the previous case where only one dimension (Ω_{dm}) was considered. That is a natural consequence of having a dataset with four times the number of instances than in the first scenario, due to the redshift inclusion.

However, if we analyse the result proportionally to the total number of PCs, both curves look similar. In both scenarios, by the time we reach around $\frac{1}{6}$ of the total number of PCs, we get around 95% of the variance explained, and by the time we reach half of the total number of PCs, practically 100% of the variance is explained.

Similarly to the previous scenario, the PCA managed to compress our initial dataset retaining most of the variance in a small proportion of the initial PCs. However, since in this case, our dataset has one

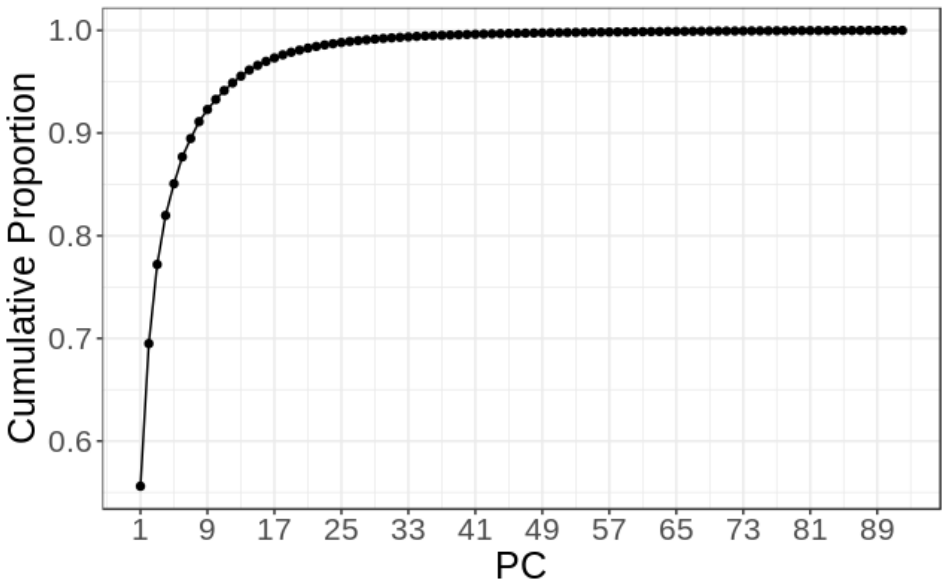

Figure 4.10: Cumulative Proportion of the Variance Explained by the Principal Components obtained through the projection of the two free-parameter (Ω_{dm},z) dataset.

additional dimension, which consequently requires the addition of multiple instances, our compression will necessarily require more coefficients.

Optimization Results

In this Subsection we analyse the optimization results. We start by considering the tree models.

The first difference to point out from the previous scenario is that we have an additional hyper-parameter, the `mtry`. This time, since we have more than one independent variable (redshift), it makes sense optimizing this hyper-parameter.

In Table 4.1.5 we can see the result for the random forest. The best results were obtained when setting `mtry` to one, which means that each time the algorithm searches the training set, it will only be randomly selecting one of our two features to be split, redshift or Ω_m. Regarding the initial grid-search on the number of trees, the best result found was `ntree=2000`, while for the `nodesize` ranges, we found the best performing one to be `nodesize=[1;30]` with Δ `nodesize =1`. Finally, the best scheme found to search over the selected range was 8-fold cross-validation.

In Table 4.1.6, we show the results for the extremely randomized trees. As opposed to the previous case, the best performing schemes were the ones where `mtry=2`, meaning that we consider both features at each training iteration. Regarding the initial grid-search on `ntree`, we found the same `ntree=2000` as in the previous case, and for the `nodesize` search range we found `nodesize=[1;25]` with Δ `nodesize=1`. Finally, the best-performing scheme was 12-fold cross-validation.

Regarding the support vector machines, we present the results in Table 4.1.7. Similarly to the one feature scenario, the best performing kernel was a radial basis one. On the other side, as opposed to that scenario, the best-found search range for both the `cost` and `gamma` parameter was the R1 range, introduced in Chapter 3. Regarding the best-performing scheme, we found it to be 10-fold cross-validation.

Finally, we show the results for NNET in Table 4.1.8. The optimization of this algorithm is by far the most computationally resourceful one, adding to that, now we are considering a dataset 4 times larger. Consequently, we decided to optimize `size` only in the ranges `size=[1;25]` and `size=[1;50]` with Δ `size=1`, as opposed to the 1 feature scenario where we extend the range to `size=75` and `size=100`. The

Table 4.1.5: Optimization results for the best found Random Forest 2D regression.

	RF
ntree	2000
nodesize	[1;30]
mtry	1
Method	8-CV
PS_r	31.17

Table 4.1.6: Optimization results for the best found Extremely Randomized Trees 2D regression.

	ET
ntree	2000
numRandomCuts	[1;25]
mtry	2
Method	12-CV
PS_r	68.84

Table 4.1.7: Optimization results for the best found Support Vector Machine 2D regression.

	SVM
Kernel	Radial
gamma	R1
cost	R1
Method	10-CV
PS_r	11.66

best-found range was `size=[1;25]` with the parameter `decay` once again, set to the default value. The best optimization scheme to search over this range was found to be 10-fold cross-validation.

Table 4.1.8: Optimization results for the best found Neural Network 2D regression.

	NNET
size	[1;25]
decay	Default
Method	10-CV
PS_r	31.14

Power Spectrum and Bispectrum Evaluation

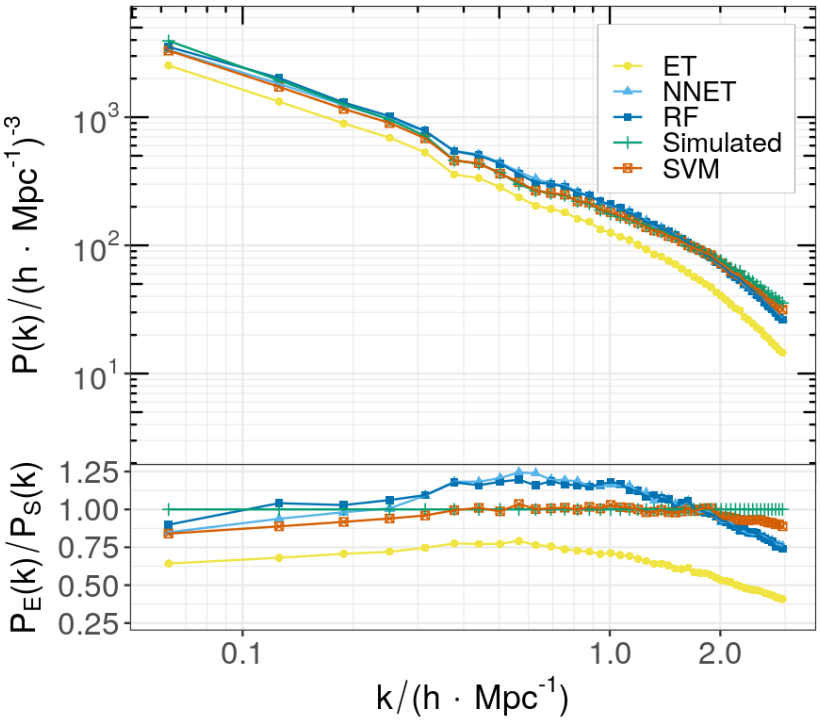

Figure 4.11: The Power Spectra of the estimated Density Field using the four algorithms on the two free-parameter (Ω_{dm},z) Dataset. **Top Panel:** Power Spectra Curves. **Bottom Panel**: Power Spectra Ratio ($P_E(k)$ = Estimated;$P_S(k)$ = Simulated).

The Power Spectrum results are presented in Figure 4.11. As expected, we have a decrease in the performance of the algorithms from the previous scenario in comparison to this one due to the increased

dimensionality of the problem. NNET is no longer the best performing algorithm, with errors reaching 25% at around $k = 0.6$ hMpc^{-1}. Both RF and NNET algorithms over-estimate the power spectrum in the range around $0.1 \leq k \leq 2$ hMpc^{-1}, while SVM and ET consistently underestimate it.

One interesting thing to notice is that, while ET and SVM seem to increase their performance at the central parts of the domain, for RF and NNET the performance decreases. The latter behaviour is unexpected since the central parts of the domain should be linked with the average properties of the density field. Consequently, it should correspond to the most representative region of the data. On the other side, we don't observe a decrease in performance on the large scale extreme, for this scenario. Instead, we notice this trend more strongly in the low scale domain, except for SVM.

Turning our attention to SVM, it is by far the best performing algorithm in this case. We managed to achieve an accuracy of more than 95% for $0.3 \leq k \leq 2$ hMpc^{-1}, decreasing to about 90% at the low scale extreme and $\sim 85\%$ at the large scale one. We can understand these results for the SVM if we recall the theoretical introduction of the algorithm. There, we stressed the fact that this algorithm is good at dealing with multi-dimensional data sets, due to its hyper-plane segregation and more particularly due to its kernel trick. So it is expected that in this higher dimension scenario, SVM obtains better results.

Finally, we should explain the abrupt decrease in the overall accuracy, between the previous scenario to the present, multidimensional one. When dealing with statistical learning, we are always subject to the so-called *Curse of Dimensionality*. It corresponds to the need to increase the size of the traing set exponentially when adding additional dimensions in the feature space, in order to obtain similar performances compared to the lower-dimensional case. In this work, due to computational constraints, when proceeding to the 2-parameter estimation, we only managed to train the algorithms with a dataset four times larger than the one we used for the 1-parameter scenario. To maintain the high degree of accuracy we saw in the previous results, we would need to further increase our training dataset.

As a final note, in Figure A.2 the results of the regressions if we neglect the optimization process of the algorithms are presented. We can see that overall, the changes are not dramatic, except for the NNET algorithm; in this case the optimization seems to be absolutely mandatory in order to achieve reasonably accurate results.

The Bispectrum results are shown in Figure 4.12. The most striking feature to notice here is that all algorithms except ET overestimate the Bispectrum, which does not happen in the one-parameter scenario.

Since ET is the only algorithm which only considers one of the independent variables, redshift or Ω_m, the other algorithms may be overestimating the Bipower due to a degeneracy between both parameters.

Indeed, in a way, redshift and matter density should have similar effects in the output of the N-body simulations. If we look at higher redshifts, we look at a past where the Universe was less dense, with a smaller amount of collapsed structures. Consequently, looking at an output at high redshift, with high matter density, can be similar to looking at an output at low redshift with low matter density.

These results further indicate that SVMs are the best method among those studied in this work, since as we can see, even looking at higher order correlations it still out-performs the remaining algorithms. It is the only algorithm able to mantain accuracies below 15% for the entire θ domain.

Finally, one additional interesting feature we see in this result, is the fact that the relation between the error and the θ domain seems to be inverted. While in previous, one dimensional case, the errors tend to increase for larger angles (corresponding to larger scales), here they tend to decrease.

This could possibly be due to the incresed size of our training dataset. Since in highest large scale extreme, what we are looking is at the whole cube, having more cubes in the dataset, in a way, means having a more representative large scale dataset. Consequently, it improves the predictions of the algorithms at such scales.

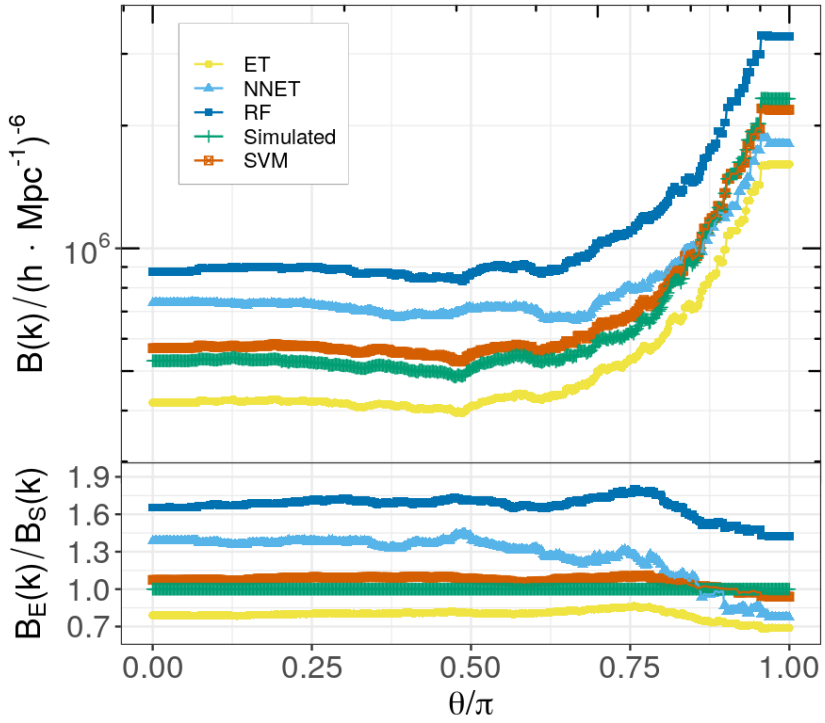

Figure 4.12: The Bispectra of the estimated Density Field using the four algorithms on the two free-parameter (Ω_{dm},z) Dataset. **Top Panel:** Bispectra Curves. **Bottom Panel**: Bispectra Ratio ($P_E(k)$ = Estimated;$P_S(k)$ = Simulated).

MOD Evaluation

Regarding the comparison through the use of the MOD metric, we show the results in Figure 4.13.

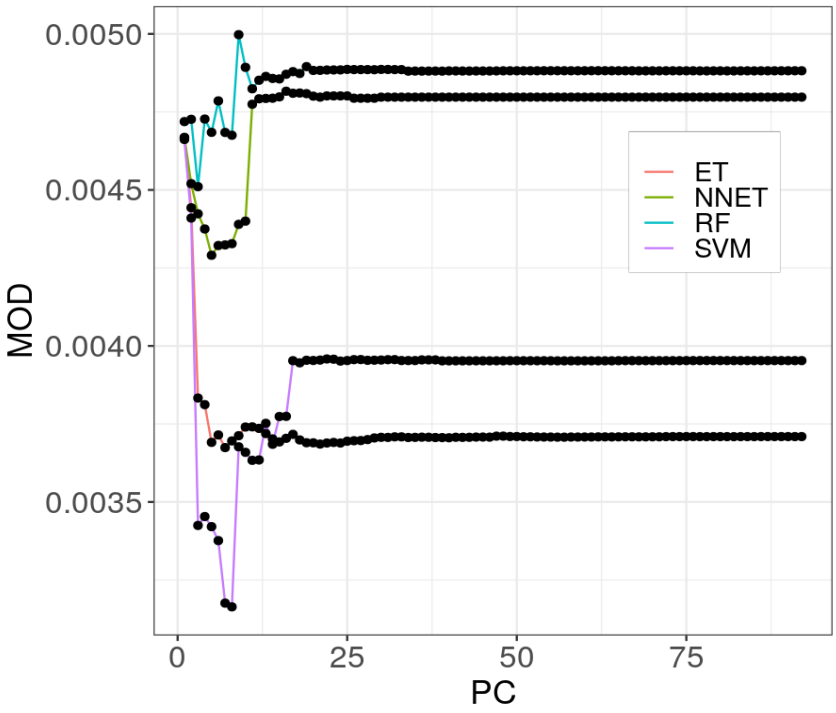

Figure 4.13: Mean Over-density Distance of the estimated density fields for the two free-parameter (Ω_{dm},z) Dataset.

Here we can see a slightly different trend form the MOD results of the one feature scenario. We see

the same increase in the NNET error as we add higher PCs, but even more accentuated for this scenario. On the other side, this time, RF and SVM change roles. In this case, RF accompanies the tendency of NNET of increasing the error through the inclusion of higher PCs, while SVM seems to be working relatively well. We also observe an increase in the SVM error for higher PCs, but less dramatic than the NNET and RF cases. Finally, we have the ET case, which seems to be quite similar to the SVM one, with the difference that ET seems to be more stable, with fewer error oscillations, for the initial PCs.

Analogously to the previous scenario, we also observe a distinct behaviour from what we see in the Power Spectrum results. Once again, this is explained by the difference in the nature of our metrics. In the MOD case, we are looking at the absolute values of the density cells. In the Power Spectrum case, we are looking at correlations in the densities at different scales. Consequently, even if the algorithms are not correctly predicting the absolute densities in the cells, they seem to be competent in predicting the statistical properties of the matter distribution, reproducing the correlations correctly.

Moreover, if we look at the MOD expression, we can see that the density values in each cell are normalized by the mean density in the cube. That means that outliers will be strongly contributing to the error. In each tail of the density distribution in our cubes, every slight error will be contributing heavily to the MOD, since we have a high discrepancy to the mean density. That could also explain the tendency for the error to increase as we add higher PCs. Higher PCs contain information on properties which highly deviate from the mean distribution and thus could be including the effects from these outliers.

4.2 Gravitational Waves

In this Section, we present the results for the Gravitational Wave parameter inference. As in the N-body context, we first show the PCA results, followed by the optimization and parameter inference results. Also similarly to the N-body case, here we divide the work into two scenarios. In the first, we apply our pipeline on Time Domain Waveforms, while in the second we focus on Frequency Domain Waveforms.

4.2.1 Time Domain Inference

Principal Component Analysis

In Figure 4.14 we can see the cumulative proporiton of the variance explained by the PCs for our Time Domain dataset. Similarly to the N-body scenario, it is possible to notice that most of the variance is held in a small percentage of the initial PCs. However, since we are dealing with a far higher amount of PCs, that small percentage will also translate in a higher number of PCs. Consequently, in this case, to obtain around 100% of the variance explained, we need at least the first 50 PCs.

Further evidence can be found if we look at the actual PC reconstruction of a waveform, shown in Figures A.3 (cumulative PC reconstruction) and A.4 (single PC reconstruction) of the Figure Appendix. Looking at the cumulative reconstruction, we can see that by the time we reach the 50th PC, adding more PCs does not produce large changes on the waveform shape. On the other side, looking at the single PC reconstruction we can see that as we project the data with higher PCs, the waveform shape gets increasingly distorted.

Optimization Results

As explained in Chapter 3, the way we tweak the S/N ratio is by defining the distance from the GW emitting system to the detector. Considering that we test our methods for six increasing distances, implying

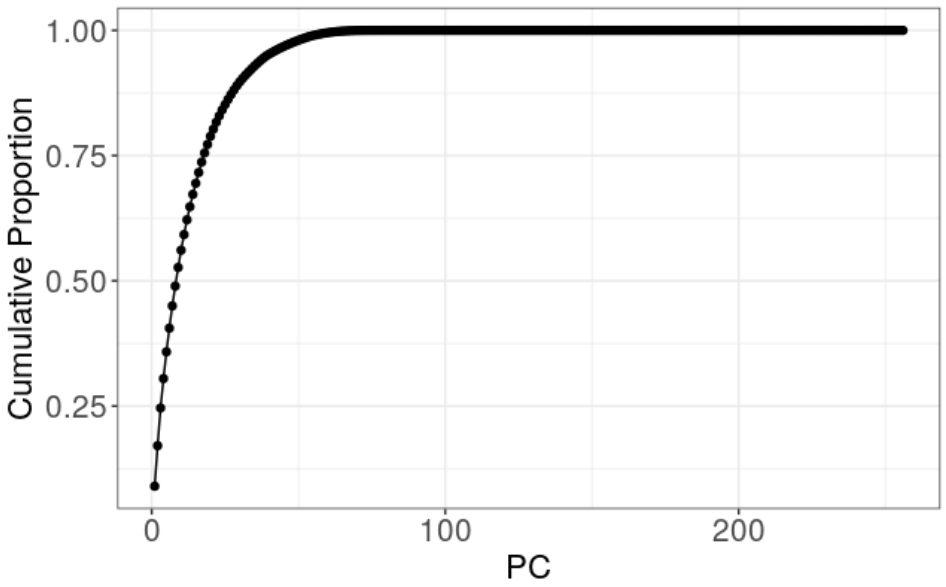

Figure 4.14: Cumulative Proportion of the Variance Explained by the Principal Components obtained through the projection of the Time Domain dataset.

six different optimization processes for each of our three ML algorithms. We show the results of the Random Forest optimization in Table 4.2.1.

Table 4.2.1: Optimization results for the best found Random Forest regressions in Time Domain, at increasing distances.

RF (TD)	**D = 1 Mpc**	**D = 25 Mpc**	**D = 50 Mpc**	**D = 100 Mpc**	**D = 500 Mpc**	**D = 1000 Mpc**
PCs	29	31	34	45	54	52
ntrees	500	1000	500	500	500	500
nodesize	Default	[1;10]	Default	Default	Default	[1;10]
mtry	seq(1,29,3)	seq(1,31,3)	seq(1,34,3)	Default	Default	Default
Method	12-CV	20-repeat Bootstrap	LOOCV	Grid-Search	Grid-Search	10-CV
RMSE	0.456	0.475	0.542	0.891	2.530	3.133

The first row of the Table shows the results for the PC optimization, the second shows the results of the Grid-Search over the number of trees and the remaining rows show the results of the optimization process on `nodesize` and `mtry`, using the `tune` function.

There are two features worthy of mentioning in this result. The first one regards the PC optimization, which seems to converge to higher numbers of PCs for waveforms with lower S/N ratio, except for the extreme case of D=1000 Mpc.

By using a higher number of PCs, the algorithms are dealing with a segment of data which contains more information on the full shape and properties of the denoised waveform, and thus their inferences should be more accurate.

The second feature that is worth mentioning is that for lower S/N ratios, the optimization schemes via the `tune` function don't seem to work, as we can see for the D=100 Mpc and D=500 Mpc cases. Once again the 1000 Mpc case seems to be an outlier in this emerging pattern.

In Table 4.2.2 we can see the results for the optimization of the Extremely Randomized Trees. Here we observe again the prevalence of the two features mentioned in the RF case. An increase of optimal Principal Component number as the S/N ratio decreases and a failure in optimization, this time for D=500 Mpc and D=1000 Mpc.

The similar results obtained from these two algorithms can be linked to their nature. As explained in

Table 4.2.2: Optimization results for the best found Extremely Randomized Trees regressions in Time Domain, at increasing distances.

ET (TD)	D = 1 Mpc	D = 25 Mpc	D = 50 Mpc	D = 100 Mpc	D = 500 Mpc	D = 1000 Mpc
PCs	12	35	37	51	67	67
ntrees	Default	Default	Default	Default	500	500
numRandomCuts	Default	Default	Default	Default	Default	Default
mtry	[1;12]	seq(1,35,3)	seq(1,37,3)	seq(1,51,5)	Default	Default
Method	10-CV	10-CV	LOOCV	12-CV	Grid-Search	Grid-Search
RMSE	0.343	0.373	0.443	0.615	1.871	2.653

previous Sections, both are tree models and we can even consider one (ET) to be a particular case of the other (RF).

Table 4.2.3: Optimization results for the best found Support Vector Machine regressions in Time Domain, at increasing distances.

SVM (TD)	D = 1 Mpc	D = 25 Mpc	D = 50 Mpc	D = 100 Mpc	D = 500 Mpc	D = 1000 Mpc
PCs	30	30	30	30	28	26
Kernel	Radial	Radial	Radial	Radial	Radial	Radial
cost	Default	Default	Default	Default	Default	Default
gamma	Default	Default	Default	Default	Default	R2
Method	None	None	None	None	None	10-CV
RMSE	0.607	0.607	0.607	0.612	0.838	1.550

Finally, in Table 4.2.3 we show the results of the SVM optimization. Here we see an unexpected break in the tendency observed in the tree models optimization.

Instead of seeing an increase in the optimal number of PCs as the S/N ratio decreases, we see the opposite. The optimal number maintains its consistency until 100 Mpc and starts to decrease for higher distances.

That indicates that SVMs should have a higher generalization capacity, compared to the Tree Models. In addition to that, we also see an unexpected failure in the optimization schemes at all distances expect, once again, for the extreme case of D=1000 Mpc, corresponding to the smaller S/N ratio.

Evaluation of the Algorithms

Here we present the main results regarding the GW chirp mass inferences. In Figure 4.15, we show the comparison between the estimated chirp masses and the ground truth at increasing distances. At first glance, we can see that the results get increasingly worse for estimations on waveforms at large distances (looking at $M_{chirp} = 16\ M_{\odot}$, we go from a $\sim 5\%$ difference at 1 Mpc to a $> 10\%$ at 1000 Mpc). If the noise remains constant for an increasingly large distance, while the strain amplitude gets increasingly smaller, consequently, the signal to noise ratio gets also increasingly smaller. That will naturally worsen the results since we perform the training with denoised waveforms.

On the other side, we can see that for distances up to 100 Mpc, the estimations are still quite good, with ratios $\sim 5\%$ for most of the domain.

One striking feature of these results is the decrease in performance, for all distances and algorithms, at the two extremes of the chirp mass domain, particularly dramatic for the lower chirp mass extreme. There are two reasons which come to mind as an explanation for this behaviour. The first, similarly to the N-body case, corresponds to the lack of sampling in the extremes.

The second explanation regards particularly the low chirp mass extreme and resides in the fact that low

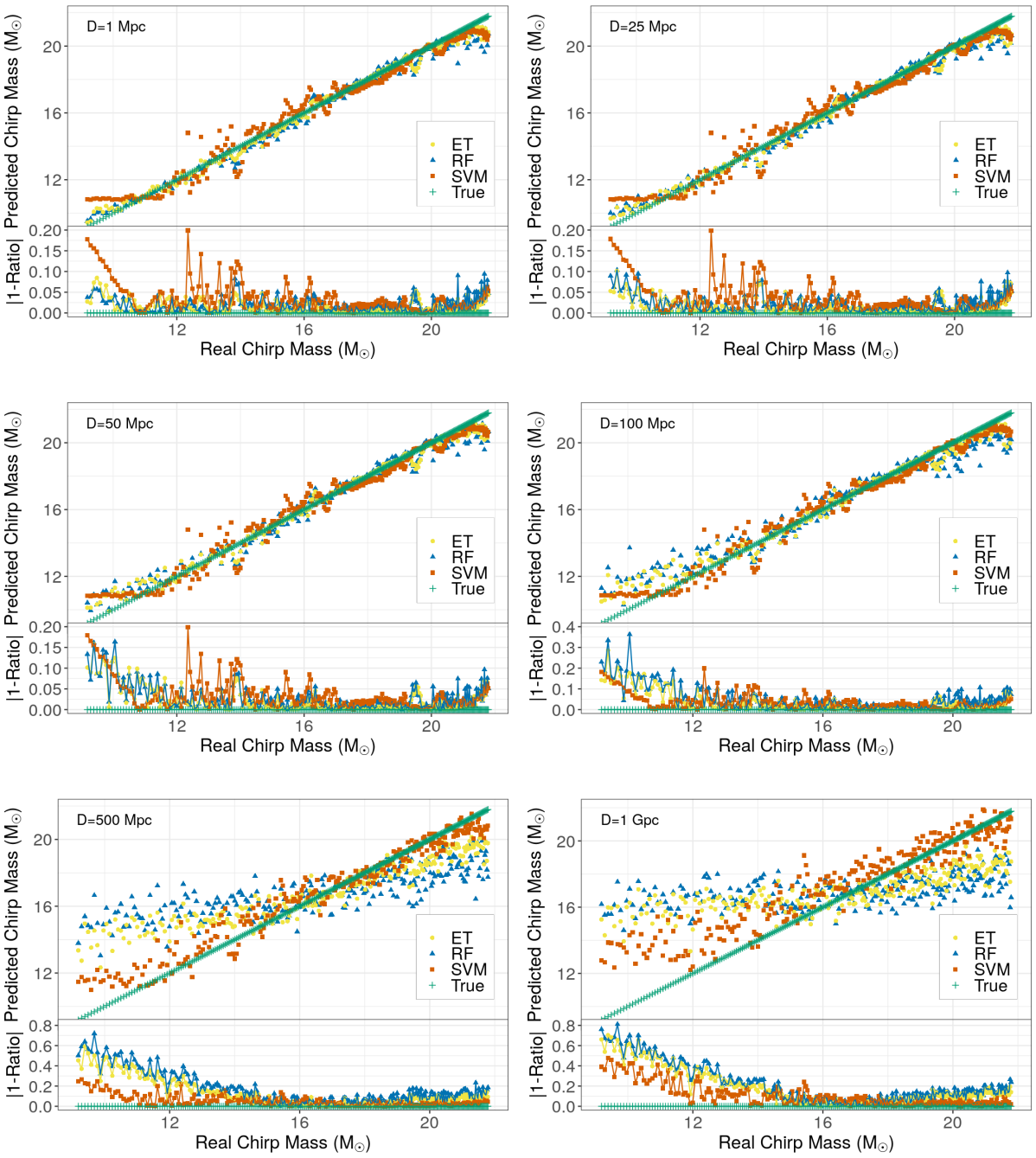

Figure 4.15: Comparison between the estimated chirp masses and the ground truth in Time Domain. The Chirp Masses were estimated using the three algorithms, on waveforms concerning GW emmiting systems at increasing distances (decreasing signal-to-noise ratios). **Top Panel:** Estimated Chirp Mass vs Ground Truth. **Bottom Panel**: Chirp Mass Ratio $\left(\left|1-\frac{Estimated}{Real}\right|\right)$. **Units:** Solar Masses ($M_\odot$).

chirp masses translate to lower strain amplitudes. Given that we have less amplitude in these cases, we will also have a lower signal to noise ratio, and consequently, these low chirp mass waveform estimations will be highly dilluted by the noise component. However, this does not explain why the results are still considerably worse in this extreme even for the D=1 Mpc case. The only explanation which comes to mind invokes the PCA projection. Since we have extremely low amplitudes in this part of the domain, these waveforms may be poorly accounted for, at least in the initial PC components that we use for the estimations.

Finally, let's compare the algorithms against each other. If we look at the first four panels, our first conclusion will be that the tree models have better performances, particularly at the discussed low chirp mass extreme, where SVM highly under-performs. Even at the more central region of the domain, SVM seems to under-perform, as we can notice by looking at the window between 12 $M_\odot$ and 16 $M_\odot$ (at all distances). On the other side, if we look at the two last panels, the situation reverses. The tree models seem to be highly sensitive to the decrease in signal to noise ratio, having much worse performances for these two cases, while SVM keeps its consistency in the prediction error and manages to be the best performing algorithm for extremely low signal to noise ratios.

As a final remark, in Figure A.7, we show the results of performing the regressions while neglecting the hyper-parameter optimization of the algorithms. There are hardly any visible changes since we are dealing with RMSE differences to the second or even third decimal place, translating in a small difference in the estimation ratio.

However, in Figure A.8, we can see a more dramatic change. We show the regression results while neglecting both the algorithms and the PC optimization, in other words, we give the full PC representation (256 PCs) in both training and testing. This better demonstrates the importance of optimizing our pipeline. This is particularly important for the SVMs, that gives a constant output, predicting a value very near the average training set chirp mass for all test waveforms.

4.2.2 Frequency Domain Inference

Pincipal Component Analysis

In Figure 4.16 we can see the cumulative proporiton of the variance explained by the PCs for the Frequency Domain. We can see that it takes a higher number of PCs to explain a given percentage of the

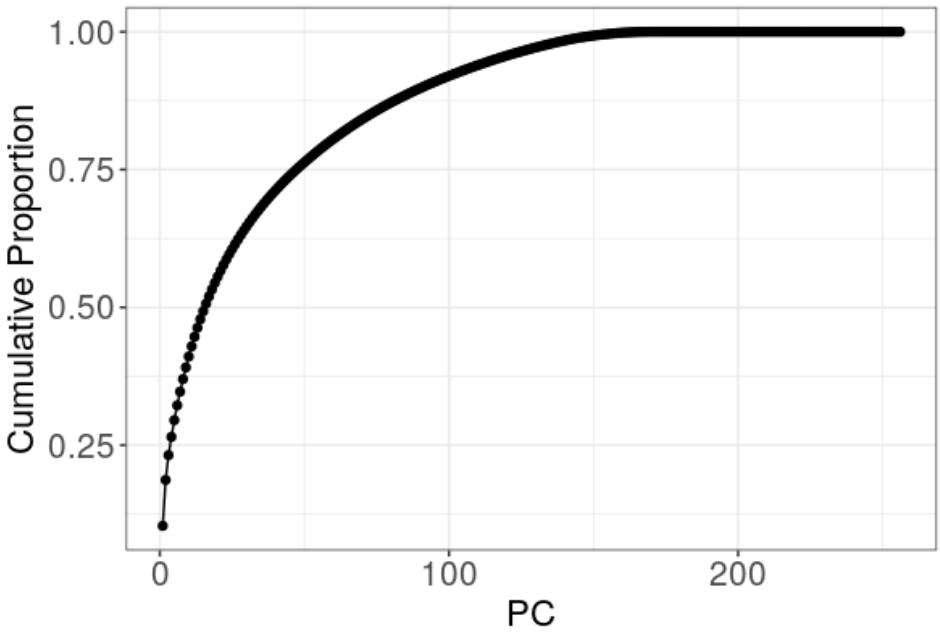

Figure 4.16: Cumulative Proportion of the Variance Explained by the Principal Components obtained through the projection of the Frequency Domain dataset.

variance in this case than in the TD case. That is most certainly related to the lower resolution of each waveform in the dataset and to the shape of the waveform itself, which seems slightly less linear in FD, as we can see in Figure 3.4.

We can also confirm these results by visualizing the cumulative and single PC FD waveform reconstructions shown in Figures A.5 and A.6 of the appendix, respectively. For the cumulative reconstruction, we only begin seeing a resemblance of the full waveform by the time we reach the 100th PC. On the other side, we see an even stranger behaviour for the single PC reconstruction, where the last PCs seem to have a somewhat random distribution.

Optimization Results

The optimization results in the Frequency Domain context are summarized in Tables 4.2.4, 4.2.5 and 4.2.6.

Table 4.2.4: Optimization results for the best found Random Forest regressions in Frequency Domain, at increasing distances.

RF (FD)	**D = 1 Mpc**	**D = 25 Mpc**	**D = 50 Mpc**	**D = 100 Mpc**	**D = 500 Mpc**	**D = 1000 Mpc**
PCs	42	42	33	33	26	2
ntree	500	500	500	500	500	500
nodesize	[1;10]	[1;10]	[1;10]	[1;10]	[1;10]	[1;10]
mtry	seq(1,42,4)	seq(1,42,4)	Default	seq(1,33,3)	seq(1,26,3)	[1;10]
Method	20-rep Bootstrap	8-CV	LOOCV	8-CV	12-CV	10-CV
RMSE	0.379	0.383	0.390	0.408	0.770	1.467

Table 4.2.5: Optimization results for the best found Extremely Randomized Trees regressions in Frequency Domain, at increasing distances.

ET (FD)	**D = 1 Mpc**	**D = 25 Mpc**	**D = 50 Mpc**	**D = 100 Mpc**	**D = 500 Mpc**	**D = 1000 Mpc**
PCs	63	73	45	45	55	2
ntree	500	500	500	500	500	500
numRandomCuts	[1;10]	Default	[1;10]	[1;10]	[1;10]	Default
mtry	seq(1,63,6)	seq(1,73,7)	(1,45,4)	seq(1,45,4)	seq(1,55,5)	Default
Method	23-CV	10-CV	23-CV	10-CV	12-CV	Grid-Search
RMSE	0.159	0.161	0.171	0.193	0.691	1.304

Table 4.2.6: Optimization results for the best found Support Vector Machine regressions in Frequency Domain, at increasing distances.

SVM (FD)	**D = 1 Mpc**	**D = 25 Mpc**	**D = 50 Mpc**	**D = 100 Mpc**	**D = 500 Mpc**	**D = 1000 Mpc**
PCs	41	41	41	41	23	14
Kernel	Radial	Radial	Radial	Radial	Radial	Radial
cost	R2	R2	Default	Default	R2	R1
gamma	R2	R2	R2	R2	R2	R1
Method	Any CV	Any CV	Any CV	Any CV	Any CV	Any CV
RMSE	0.363	0.372	0.376	0.394	0.665	0.925

Compared to the estimations in Time Domain, we can clearly see an improvement in the results, for all algorithms. This reason can be possibly attributed, again, to the lower resolution of this dataset. On one side, makes it harder for PCA to include a maximal amount of variance in a minimal amount of PCs. On the other, it seems that the PC to Chirp Mass relation becomes more predictable.

One interesting feature we can also see in these results is that fact that this time the number of optimal PCs seems to increase with decreasing S/N noise, except for SVMs, in opposition to the Time Domain case. This indicates that higher PCs contain crucial information to disentangle the noise from the true shape of the waveform.

Here we remark that the optimization works for the lowest S/N ratios, which is expected since the true waveform shape does not get as diluted in the noise as in the case of Time Domain waveforms (see Figures 3.5 and 3.6). The same reason justify the increase in the performance of the algorithms for D=500 Mpc and D=1000 Mpc in this case.

Finally, we can also see the SVM breaking the tendency in results compared to the remaining algorithms, both in the relationship between the optimal number of PCs and the S/N ratio and also in its high performance for low S/N ratios.

Evaluation of the Algorithms

In Figure 4.17 we can see our main results in Frequency Domain, comparing the estimated chirp masses with the ground truth. In general, we can see the same features that are present in the Time Domain estimation context. The results also show a general tendency of decrease in performance for higher distances (lower S/N ratios) and at the extremes of the chirp mass domain. Once again, this decrease is particularly accentuated at the lower chirp mass extreme.

Regarding the comparison between the algorithms, we also see a similar behaviour as the results in the previous Sections. The tree models are the best performing algorithms for data with high S/N ratio and the SVM seeming almost unfazed in its estimation error for low S/N ratios, surpassing the tree models in performance for the cases of estimations at D=500 Mpc and D=1000 Mpc.

It is in the general performance of the algorithms that we see the main difference. In this context, the results improve considerably compared to the Time Domain analysis. If we focus on the higher S/N ratio side (up to 100 Mpc), we see that while in TD the predictions fall in the 5% to 10% difference (at the central parts of the domain), in FD they fall consistently in the 2% to 5% difference. If we look at the lower S/N ratio results, in TD we see that generally, the results fall in the 10% to 20% difference while in FD we have results oscillating between a 5% to 10% difference.

Another interesting thing to notice is the fact that the lower chirp mass extreme where the results start to worsen considerably seems to be shifted to much lower chirp masses in FD than in TD, meaning that we have a much smaller range of lower extreme misbehaviour in the FD estimation context. The results in D=500 Mpc exemplify this. In this case, while in the TD estimation context the lower chirp mass estimations start to considerably diverge from the ground truth at around $M_{chirp} = 14\ M_{\odot}$, in the FD case, this turn over seems to only occur around $M_{chirp} = 12\ M_{\odot}$.

It is interesting to notice that although PCA has a higher difficulty compressing the data in the FD case than in the TD case, the actual supervised algorithms estimations are better. Both behaviours could be due to the same reason, and here we again invoke the lower resolution of the dataset. On one side, the lack of single instance information gives PCA a hard time compressing the data into a lower amount of PCs. On the other side, the final compression coefficients seem to be quite more predictable than ones found in the TD PCA, giving the ML algorithms an easier job in building their regression models.

Once again, as a final remark we show the unoptimized versions of our regressors in Figures A.9 (Default Algorithms) and A.10 (Default Algorithms and PCs). We can see similar results to the TD case. Regarding the optimization of the algorithms, the differences are subtle. It is the PC optimization which introduces dramatic improvements to the picture. However, we should not disregard the optimization of the algorithms. Even if there are no nitid changes at the naked eye, since at the central parts of the domain we are able to obtain ratios down to around 1% and 2%, every slight improvement is relevant.

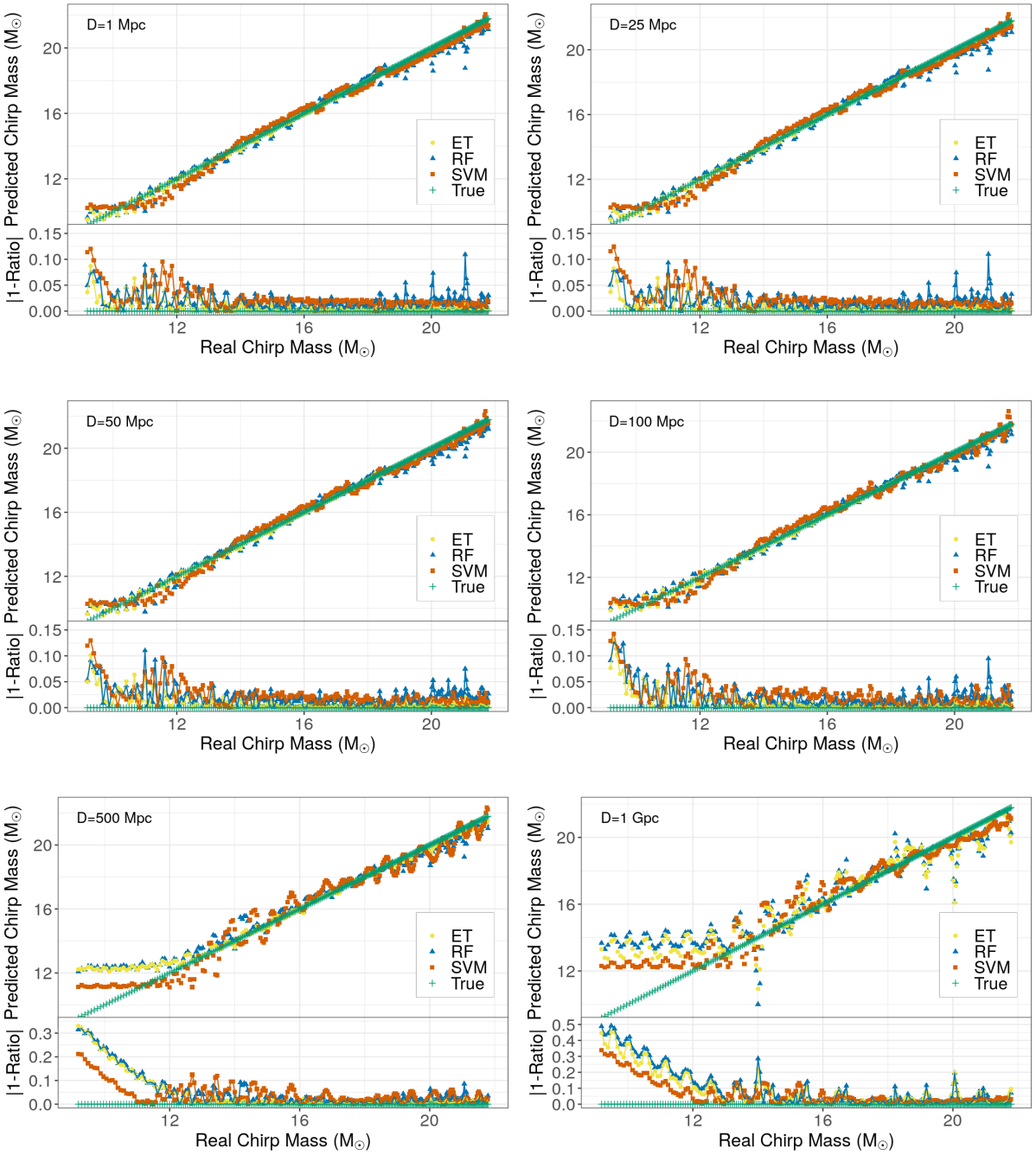

Figure 4.17: Comparison between the estimated chirp masses and the ground truth in Frequency Domain. The Chirp Masses were estimated using the three algorithms, on waveforms concerning GW emmiting systems at increasing distances (decreasing signal-to-noise ratios). **Top Panel:** Estimated Chirp Mass vs Ground Truth. **Bottom Panel**: Chirp Mass Ratio $\left(\left|1-\frac{Estimated}{Real}\right|\right)$. **Units:** Solar Masses ($M_\odot$).

Chapter 5

Discussion and Future

As described in the previous Chapters, this thesis proposes a new method to address well-known problems in Cosmology and Astrophysics. We showed that combining PCA with simple supervised learning methods allows fast and accurate estimation (regression) of cosmological density fields. We also demonstrated that the same method can be used, in the inverse way, to perform parameter inference, applied to the problem of GW source parameter determination.

Indeed, the ever-increasing need for larger and more diverse sets of cosmological simulations, to study large-scale structure mechanisms and their dependence on Cosmology, demands fast and accurate methods to produce multiple realizations of density volumes. Our results show that our supervised learning method innovatively achieves this objective. On the other hand, our results demonstrate that our proposed method also provides a fast way of performing GW parameter inference without requiring additional prior assumptions on top of those that were already adopted to build the set of approximated waveforms.

In this chapter, we discuss the main results of our work. We comment on the relevance of our method for further developments in these fields, addressing its gains of performance and potential to explore other situations of interest. We also address the method's present constraints and discuss ways to mitigate them, proposing solutions for future implementation.

5.1 Efficiency

Here we focus on the efficiency of our method. In other words, we will discuss the amount of CPU time and computational resources required to produce a final result. We show how much time it takes to run each main segment of the pipeline, and compare the total time to perform the estimations with the time taken to perform the N-body simulations.

As mentioned at the beginning of Chapter 3, our pipelines were implemented using an Intel Core i5-4460 with a 16GB RAM of 2400 Mhz, and no discrete GPUs

5.1.1 Cosmological Density Fields

Regarding our work in N-body Density Fields, in Table 5.1.1 we show the CPU running times for our two estimation scenarios, divided into the five main segments of our pipeline.

In the first row, we have the case of single free-parameter estimation, where we present the running times of our best performing algorithm at $z = 0$, the NNET estimator. In the second row, we see the times for our two-parameter estimation scenario, where we present again the times for our best regression, this time being the SVM estimator.

Overall we can see that our second scenario takes quite more time to run, as it should since we are dealing with a larger dataset: the Ω_{dm} case needs to deal with a 385 MB dataset while the (Ω_{dm}, z) needs to deal with 1.5 GB. The difference being especially striking in the PCA projection and consequent

Table 5.1.1: CPU Running times for each main segment of the N-body Density Field estimation pipeline. First row concerns the best-found 1D regression using NNET at $z = 0$. Second row concerns the best-found 2D regression using SVM at $z = 0$.

Density Fields	Loading & Treatment	PCA	Optimized Reg.	PC Estimation	De-Projection into Original Basis	Total
Ω_{dm} **(NNET)**	2.606s	20.926s	3min27s	0.077s	0.719s	**~3min51s**
(Ω_{dm},z) (SVM)	9.458s	1min58s	5min02s	0.082s	4.380s	**~7min14s**

de-projection where for the latter we have an order of magnitude difference. We also confirm that the optimization process is the lengthiest segment of the pipeline, as we can observe for both cases.

However, the three first segments of the pipeline (Loading & Treatment, PCA and Optimizations) only require a single execution. Once trained and optimized, the algorithms do not need to be trained again in order to perform additional estimations, without extrapolation. Consequently, the only segments of the pipeline which should be considered to gauge the efficiency of our work against a full N-body simulation, and also to be compared against the literature, are the two last segments (PC estimation and De-projection).

We show the total time in order to perform the emulations, in Table 5.1.2

Table 5.1.2: CPU Running times for emulation segment of the N-body Density Field estimation pipeline.

Emulation	PC Estimation	De-Projection	Total Emulation Time
Ω_{dm} **(NNET)**	0.077s	0.719s	**0.796s**
(Ω_{dm},z) (SVM)	0.082s	4.380s	**4.462s**

We consider these results as highly successful, considering that it takes $\sim$ 84 min to perform an N-body simulation of the same scale. We managed to achieve high accuracies in a density field estimation, which takes a total of $\sim$ 0.796 s for the one-dimensional case and $\sim$ 4.462 s for the two-dimensional case. That corresponds to a three orders of magnitude difference in the time required. Moreover, the computational resources required to implement our pipeline are far less than those required to perform the N-body simulations.

In a nutshell, we managed to achieve an outstanding result regarding the trade-off between efficiency and accuracy in our methods. We managed to emulate the density fields reproducing the real Power Spectra to less than a 1% differences (NNET in 4.5 at $0.4 \leq k \leq 3$ and SVM in 4.11 at $0.4 \leq k \leq 2$), while at the same time dramatically minimizing the computational requirements and time required.

5.1.2 Gravitational Waves

In this Subsection, we focus on our work on Gravitational Waves. We present the pipeline CPU running times in Table 5.1.3.

In the Gravitational Wave case, we also analysed the PC optimization, and this is indicated in Table 5.1.3. Together with the optimization of the algorithms, PC optimization is one of the lengthiest parts of the pipeline. Moreover, to capture both extremes of S/N ratio we present times for the best regressors at each extreme of the distance domain, D=1 Mpc (ET) and D=1000 Mpc (SVM), for both Time Domain (TD) and Frequency Domain (FD) regressions.

Some results presented in Table 4.2.5 are only understood if we keep in mind that since we include the optimization in the total time, it is natural that some ranges take a much longer time than others. This

Table 5.1.3: CPU Running times for each main segment of the GW inference pipeline. First two rows concern Time Domain inferences, in waveforms at D=1 Mpc and D=1000 Mpc, respectively. Last two rows concern Frequency Domain inferences, for the same distances.

GWs	Loading & Treatment	PCA	PC optimization	Optimized Reg.	Project New Waveforms into PC basis	Infer Chirp Mass	Total
TD D=1 Mpc (ET)	21.867s	0.984s	7min29s	13.438s	1.562s	0.034s	**~8min07s**
TD D=1000 Mpc (SVM)	5.651s	0.985s	6min03s	1.760s	1.570s	0.008s	**~6min13s**
FD D=1 Mpc (ET)	6.862s	0.168s	2min59s	44min8s	0.142s	0.041s	**~47min16s**
FD D=1000 Mpc (SVM)	7.228s	0.187s	1min10s	9.741s	0.252s	0.004s	**~1min28s**

is the case of FD D=1 Mpc regression, where the best optimization range found was a quite extensive one. If we look the results for this case, we can see that for D=1 Mpc the best result was the simultaneous optimization of both the `mtry` and `numRandomCuts` parameter, using the most time-consuming optimization scheme, 23-fold CV. Together with the fact that we need to search a relatively large range of `mtry` values, corresponding to 63 PCs, these factors contribute to the long, and at a first sight apparently abnormal, $\sim 47m$ min optimization run time.

On the other hand, if we look at the D=1 Mpc TD results, we see that it takes a significantly smaller amount of time for ET to perform the optimized regression (13.438 s). Three reasons may justify this result. First, for D=1 Mpc, as we can see in Table 4.2.2, we have a much smaller number of optimal PCs (twelve) than in the FD case (sixty-three). Second, the optimal range found for TD was the single optimization of the parameter `mtry`, while `numRandomCuts` was set to its default value. Third, the scheme takes less time to run since we are dealing with 10-fold CV instead of 23-fold CV.

If we consider the remaining segments of the pipeline, the time domain pipeline tends to take a longer time to run than the frequency domain pipeline. And this stems mainly from the loading and treatment of the data, as well as the PCA. Since the TD waveforms have approximately 6 times more data points, our initial training data matrix will be considerably larger than in the FD case, and consequently, it will take more time to load, treat and perform the basis projection of this data.

If we look at the particular case of low S/N ratios (D=1000 Mpc), it is appealing to see that we have a much more efficient pipeline, with a total time of 6min13s for the TD case and 1min28s for the FD case. These improved results stem from the fact we use SVM for low S/N ratio, which is faster than the other algorithms, both in its PC and hyper-parameter optimization.

Finally, in Table 5.1.4, we show the total time of the inference process, which concerns the last two segments of our pipeline. Once again, we achieve a highly significant efficiency in inferring the Chirp Mass of the waveforms. We manage to obtain inference times down to 1.578 s for Time Domain inferences, and down to 0.183 s for the more realistic case of Frequency Domain inference. These results are particularly appealing, considering that we are dealing with 255 simultaneous M_{chirp} inferences.

Table 5.1.4: CPU Running times for the inference segment of the GW pipeline.

Inference	**Projection into PC basis**	**Inference**	**Total Time**
TD D=1 Mpc (ET)	1.562s	0.034s	**1.596s**
TD D=1000 Mpc (SVM)	1.570s	0.008s	**1.578s**
FD D=1 Mpc (ET)	0.142s	0.041s	**0.183s**
FD D=1000 Mpc (SVM)	0.252s	0.004s	**0.256s**

5.2 The Supervised Learning Algorithms

Regarding the supervised learning algorithms used in this work, there are some important remarks to be discussed. We explore them in this Subsection.

Let's first focus on the tree models. When we look at our application in the N-body context, compared to SVM and NNET, we can say that they under-performed, especially ET which got particularly bad results, as we can see in Tables 4.1.2 and 4.1.6. On the other side, for the Gravitational Wave case and if we consider inferences from waveforms with high S/N ratios, for $D < 100$ Mpc, the tree models perform quite well. This time ET consistently performs better than RF both in Time and Frequency Domain, as we can see in Tables 4.2.2 and 4.2.5.

That indicates that the random splitting process done by ET works better in the PC space than in the Ω_{dm} and z spaces since in the GW context, we are using the PCs as features. Consequently, if we consider that ET has the advantage of consuming less time and computational resources than RF, we can conclude that in this context it is definitely the appropriate choice, over RF, as opposed to the N-body context.

Nevertheless, it is NNET and SVM which stand out in their results for both contexts. NNET manages to achieve the best results for the one dimensional N-body regression case that we observe in Table 4.1.3 and Figure 4.5. SVM manages to outperform all the other algorithms in multi-dimensional (N-body) and low S/N ratio (GWs) contexts.

Considering that our current cosmological models pertain a multitude of cosmological parameters not included in this work (e.g Hubble constant, Baryonic Matter Density, spectral index), in a more realistic multi-dimensional ($D > 2$) context, SVM should stand out as a favourable method. Moreover, if we focus on the GW inference problem, in real detector waveforms always come imbued in a significant amount of noise, consequently making SVM stand out as a good methodological choice for analysis pipelines.

Nevertheless, these results do not allow a final conclusion to be reached with respect to the potential of both algorithms. It is important to keep in mind that our optimization process does not necessarily achieve a global minimum in our regression error metrics and algorithm cost functions. Since we searched a finite amount of hyper-parameter values and ranges, there is no guarantee that the algorithms were not driven to a local minimum with further room for improvement by searching other segments of the hyper-parameter space. Our results, however, are a strong indication of the potential of the algorithms that were analysed.

Moreover, looking at the particular case of the NNET, and at its under-performance for high dimensional cases, this was an expected result, not implying a deficiency of the algorithm in applications in these contexts, but of the adopted architecture of the network. The results obtained here were a conse-

quence of the simplicity of the single-layer architecture that was adopted. To deal with non-linear and high dimensional data there is a need to adapt the NNET architecture accordingly, and to rely on building multi-layer, deeper, models.

5.3 Comparison with Similar Work

Our proposed method includes a few innovations regarding the present literature landscape in this area.

On the side of the N-body density fields, our work stands as a probe on the possibility of performing full 3D dark matter density field emulations with high efficiency and accuracy, given a set of cosmological parameters and using the simplest supervised Machine Learning tools while relying on very small training sets and relatively modest computational resources.

Such simple ML tools are usually used in the literature to perform parameter estimations. In the case of [67], they use Support Vector Machine and K-Nearest Neighbours to map Dark Matter halo properties to the number of galaxies expected in the halo. In other words, they predict the number of galaxies given a set of Halo properties. Work on Dark Matter Halo properties with Random Forest is performed in the case of [68], where this time, they predict if a set of N-body dark matter particles belongs to a halo in a given mass range. In [69] and [70], a multi-layer neural network is implemented in order to predict the Power Spectrum of an N-body simulation given a set of cosmological parameters.

More recently, there has been a cosmological community shift to a paradigm of Deep Learning Neural Networks models. These highly computer-intensive tools are used to perform more challenging tasks, such as inference of parameters from full N-body simulations and also the emulation of the simulations themselves – although this was not yet the case when our work started. In [71] and [72], Convolutional Neural Networks (CNNs) are applied to estimate cosmological parameters such as Ω_{dm} and σ_8 from full 3D N-body simulations. The latter work used Tensorflow [73] parallel processing abilities, which enable the segmentation of the training process through different Graphical Processing Units (GPUs) and different CPUs. In [74] Generative Adversarial Networks (GANs) are implemented and trained on a dataset of 2D N-body slices for a given cosmology, to learn the probability distribution of such dataset and consequently, emulate different realizations of the same cosmology. In [75], they conduct similar work, but this time using GANs on a 3D dataset. Another interesting and related work is the case of [76], where they use a GAN to perform emulations of 2D dark matter slices, followed by the construction of a Variational Auto-Encoder (VAE), used to perform estimations of the cosmological parameters associated with the emulated fields. A different approach is implemented in [77], where they use the displacement field vector obtained from the Zel'dovich approximation to train a CNN to emulate the actual evolved density field (using FastPM approximations [78]), correspondent to the provided displacements. Since different displacement fields can produce identical density fields, this method ensures the removal of the ambiguity. Moreover, it short-cuts the need to train on the initial random seed provided in the initial conditions of the simulation. Finally, and closer to our work, in [79], they improve the work of [77], using full N-body simulations, instead of FastPM approximations.

As it is possible to notice from this short overview of machine learning techniques applied to N-body simulations, most of the ML landscape used for density field emulation is populated with Deep Models. Moreover, even when training on full 3D distributions, most of these works downsample the simulations to provide them to the algorithms without computational bottlenecks, as a dimensionality reduction technique. Additionally, few cases are found where the emulations of full N-body simulations can be obtained for a given set of free cosmological parameters.

To our knowledge, our work is a first successful attempt at generating full 3D N-body simulations

for a given set of cosmological parameters, while adopting simpler Machine Learning Models, which are more amenable to future interpretation. The main enabler of this contribution is naturally the use of compression of dimensionality reduction step, here a PCA. The fact that large scale structure contains a high degree of homogeneity and isotropy implies that it is possible to profit from these properties to simplify the estimation of density cubes. By finding good transformations, invariances (rotational and translational) can be exploited to find a compressed representation ultimately enabling us to provide full N-body density fields to the simplest machine learning models. Here the PCA provided a first approximation to such transformation, but further compression methods as MOPED [80, 81] can certainly result in even more interesting results.

On the side of the Gravitational Waves, a large number of studies involving Machine Learning have also been produced. Going back to 1999, in [82] a Multi-Layer Neural Network-based model was applied to identify noise in theoretical VIRGO signals. Since then, many studies have come forth, for instance in glitch detection using Support Vector Machines, Random Forest and Neural Networks [83, 84] and in the detection of GW signals associated with gamma-ray bursts using Decision Trees [85] and Neural Networks [86].

More recently, and similarly to the N-body case, Deep Learning has also become the main methodological focus. Deep Learning was adopted in the pioneering work of [87] to detect BBH (non-spinning BBHs on quasi-circular orbits) GWs as a substitute to Matched Filtering and to perform parameter inference, in simulated LIGO waveforms, and shortly after, on real waveforms [88]. More recently, works on modelling the posterior distribution of the GW parameters with Deep Models have also been performed in [89–92], in both simulated and real waveforms. Finally, work on Gravitational Wave emulation, similar to what we did in the present thesis to 3D density fields, has been recently performed [93].

Our work in this context can be seen, once again, as a demonstration of the viability in applying simple supervised learning tools to perform parameter inference on compressed waveforms. In a way, this work can be seen as a generalization of our N-body density field pipeline, to different astrophysical/cosmological contexts involving gravitational dynamics, and different estimation contexts (emulation and inference) involving Machine Learning Tools.

5.4 Future Applications

Here we discuss possible research avenues for the future of this work. We begin by making remarks on possible new approaches to be explored, and following that we discuss how we can improve our current pipeline.

5.4.1 Exploring New Approaches

Cosmological Density Fields

On the side of the Cosmological Fields, there is still a clear improvement to be made, which regards the initial conditions of the N-body simulations. In particular, the seed for random structure generation. In the case of our work, we deal with simulations having a fixed initial condition seed. That means that the structures are generated with fixed spatial configurations. That is one of the reasons for the use of the correlation function as our accuracy/algorithm performance metric, given that it is independent on the initial simulation seed and concerns only the statistical properties of the field.

We could further improve our methods by generating structures of different spatial distributions, while preserving the same statistical properties, for a fixed set of cosmological parameters. One way to do it

would be to use the actual N-body initial displacement field in the initial training dataset, instead of the direct output density vector for a certain time step. As mentioned before, this approach was already tested successfully in [72] and [75].

An additional improvement would be to attempt an expansion of the cosmological parameter space, in the same lines of what we made by including the redshift in the second estimation scenario, but with additional relevant parameters. Of course, to achieve that a more developed application of the methods should be made, and possibly even the exploration of Deep Neural Networks, which are more appropriate for high dimensional contexts. Of course, due to the curse of dimensionality, we would need to increase the data in our training dataset considerably.

Additionally, there is also the possibility of adapting this methodology for N-body simulations with hydrodynamics, or in other words, gas. That would mean the inclusion of all the gas physics required in the simulations and a proper adaptation of the ML methodology, which if successful would bring this work closer to the reality we observe, providing us with means to probe how the inclusion of baryons impacts the power spectrum/bispectrum of the fields. Moreover, another possibility would be the inclusion of the methods in the actual code of the N-body simulation, to internally accelerate the N-body field generation process.

Lastly, we could also apply our methodology in order to generate large sets of N-body realizations around a given fiducial cosmological parameter value, which can then be used to forecast and constrain the parameters through statistical analysis.

Gravitational Waves

Focusing now on the Gravitational Wave applications, there are also some clear routes to follow.

Similarly to the N-body case, one route would be the increase of the astrophysical parameter space. Adding to the Chirp Mass, we could introduce relevant parameters such as the spin of the Black Holes, which would also demand an increase in the training dataset size.

Another possible avenue is the employment of such methods in waveforms computed for other physical contexts, such as Modified Gravity and other relevant models to be tested. Currently, the PyCBC platform does not provide Modified Gravity waveforms, and thus other avenues would need to be searched.

One interesting approach would be to emulate the Gravitational Waves, given a training dataset of Numerical Relativity waveforms, instead of approximants. This way we could perform similar work to [89] where they train their parameter inference methods on a dataset of emulated waves, provided by their Machine Learning method.

Finally, we also have the obvious route of experimenting with real data, using the LIGO observations data, also available through the PyCBC platform. We could attempt to apply our current pipeline directly on these waveforms, or improve it by including waveforms already imbued in noise in the training set, at the cost of decreasing the generalization capacity of our algorithms.

5.4.2 Improvements to the Pipeline

There are two obvious ways to improve our pipeline without changing its main framework (PCR).

One way is through the improvement of the optimization pipeline. That could be achieved simply through the enlargement of the hyper-parameter search ranges. In a more sophisticated way, we could also write our own optimization code, and force it to internally account for our performance metrics. One possibility to experiment would also be to try and perform optimization via a stochastic search of the hyper-parameter space. In this case, instead of a typical deterministic search over a regular grid of

hyper-parameters, we randomly chose the initial values, set a performance metric, stopping criterion, and randomly search over the parameter space until the chosen criterion is fulfilled.

Extending the hyper-parameter search range is a straightforward way to improve the optimization since due to time and memory constraints, most of our searches were in quite limited ranges.

Additionally, instead of using the available R optimization packages and connecting their results with an external performance metric, we could build our own code already accounting internally for those performance metrics. For the N-body case, we could optimize the algorithms using additionally the bispectrum as a reference metric. That would certainly improve the estimations of the properties probed by the latter metric.

Finally, one obvious and simple way to improve our pipeline is through the increase of the training dataset. By exponentially increasing the data (given available resources), we should be able to obtain much better results in the dimensional contexts presented in this work, and we should also be able to include additional dimensions, translating in additional relevant parameters.

Chapter 6

Conclusion

In this thesis, we presented a methodology to enable faster sampling and inference of datasets in two gravity dominated contexts, Structure Formation and Gravitational Waves. To achieve that we implemented supervised Machine Learning methods coupled with Principal Component Analysis, on training datasets of full 3D N-body simulations and approximated Gravitational Waveforms.

In the context of structure formation, we provided contributions to the field using our methodology for fast and accurate emulations of full N-body dark matter density fields of 128^3 density cells in a box of 100 h^{-1}Mpc, given a choice of cosmological parameters. We applied four supervised Machine Learning Algorithms, Random Forest, Extremely Randomized Trees, Support Vector Machine and Neural Networks, in two different emulation scenarios. In the first scenario, we performed emulations solely based on the Dark Matter Density (Ω_{dm}) parameter. We managed to reproduce the Power Spectrum of the corresponding simulated field to a less than a 1% difference for $k > 0.3$ hMpc^{-1} in all redshifts, and reproducing the Bispectrum for all θ domain to less than a 3% difference for $z = 0$ and less than a 2% difference for the remaining redshifts, using the best-performing algorithm (Neural Networks). We additionally provided the first successful adoption of Functional Principal Component Analysis in the context of structure formation. We achieved even better results than the supervised methods for a reduced size dataset of simulations with 64^3 density cells, managing to reproduce the Power Spectrum and Bispectrum to less than a 2% and 5% difference from the ground truth, at all redshifts. In this scenario we trained our algorithms with 23 N-body simulations in the range $\Omega_{dm} = [0.05; 0.6]$ with $\Delta\Omega_{dm} = 0.025$, and tested the emulations for $\Omega_{dm} = 0.309$.

In the second scenario, we introduced an additional dimension, adding the redshift as free-parameter. Using the best-performing algorithm in this scenario (Support Vector Machine), we managed to reproduce the Power Spectrum to less than a 5% difference for $0.3 \leq k \leq 2$ hMpc^{-1} and the Bispectrum to less than a 15% difference in all θ domain. In this scenario, we trained the algorithms with simulations in the same Ω_{dm} range given in the previous one but in four distinct redshift snapshots ($z = 0$, $z = 0.25$, $z = 1$, $z = 10$), increasing the training dataset by a factor of four. We tested the emulations against an $\Omega_{dm} = 0.309$ simulation at redshift $z = 0.5$.

We consider this a success considering that for percent level accuracies, we managed to perform the emulations in 0.796 s in the first scenario and in 4.462 s for the second scenario, which corresponds to a three orders of magnitude difference compared to the time required to perform an N-body simulation of the same scale.

In the context of Gravitational Waves, we inverted the pipeline to achieve the aim of inferring the Chirp Masses for a set of Time and Frequency Domain simulated waveforms. We managed to achieve a high generalization capacity in our models by training them with 256 denoised waveforms, and testing on 255 waveforms imbued in theoretical LIGO detector noise, for six different datasets at decreasing signal-to-noise ratios. In Time Domain, we managed to perform the inferences to around a 5% difference for the higher signal-to-noise contexts and $M_{chirp} > 12\ M_{\odot}$ and to around 10% difference in the lower signal-to-noise ratio contexts and for $M_{chirp} > 16\ M_{\odot}$, using the best-performing algorithms. For both

high and low signal-to-noise ratios, it took less than 2 seconds to perform the 255 inferences.

In Frequency Domain we achieved better results, obtaining differences in the inferences of around 2% for the higher signal-to-noise contexts and for $M_{chirp} > 14\ M_{\odot}$, and of around 5% for lower signal-to-noise ratio contexts and for $M_{chirp} > 16\ M_{\odot}$. In this case, for both high and low signal-to-noise ratios, it took less than 0.3 seconds to perform the 255 inferences.

The main success in the Gravitational Wave parameter inference results resides in the generalization capabilities of both the algorithms and our overall methodology/pipeline. The algorithms, in making reasonably accurate inferences in noisy datasets, while being trained on denoised ones. The pipeline, is showing outstanding adaptability from a context of Emulation to a context of Parameter Inference, in quite distinct datasets.

The layout of our thesis comprised the following parts.

In Chapter 1, we introduced the theoretical concepts relevant for a better understanding of our work. In the context of N-body simulations, we began by introducing the Λ-*CDM* model, followed by a brief contextualization in the subject of Structure Formation and the definition of statistical quantities such as the Power Spectrum and Bispectrum. Lastly, we gave a brief explanation of the inner working of the simulations. In the context of Gravitational Waves, we begin by introducing the linearized theory of gravity, followed by a short overview on the generation of Gravitational Waves from Binary Sources, and concluded by making remarks on the waveform characteristics and the available methods to compute it. In Chapter 2 we introduced the theory behind the Machine Learning Algorithms as well as the optimization methods, giving brief details on the mathematical formulation of the models, and explaining the distinct optimization approaches considered. In Chapter 3 we described the methodology of our work. We provided details on the implementation of our pipeline, in both scientific contexts, explaining the processes of dataset generation, PCA compression, Machine Learning algorithm training, optimization, and testing. In Chapter 4 we presented the results. Regarding the N-body simulations, we divided our work into two scenarios, emulation from a single free-parameter (Ω_{dm}), and emulation from two free-parameters (Ω_{dm},z). For the Gravitational Wave inference pipeline, our work was similarly divided into two scenarios, Chirp Mass inference from Time Domain waveforms, and Chirp Mass inference from Frequency Domain waveforms. For both scientific contexts, we began by showing results concerning the PCA compression, followed by the Optimization results. Regarding the evaluation of the models, for N-body Emulations, we presented comparisons with ground truth using both the Power Spectrum and Bispectrum as well as a quantity which we defined as the *Mean Over-Density Distance* (MOD). For the Gravitational Wave scenario, we evaluated the parameter inferences against the ground truth using the *Root Mean Squared Error* (RMSE). Lastly, we also showed a demonstration of the application of Functional Principal Component Analysis in the context of single free-parameter N-body emulation. Finally, in Chapter 5, we discussed the main findings and future of our work. We began by discussing the efficiency of our pipelines, followed by a description of the relevant features we found concerning the statistical learning methods performances. We proceeded to give a brief review of our work as compared with the literature. Lastly, we gave remarks on possible future paths, both in the exploration of new scientific contexts and also in the improvement of this work's pipeline.

Appendix A

Figure Appendix

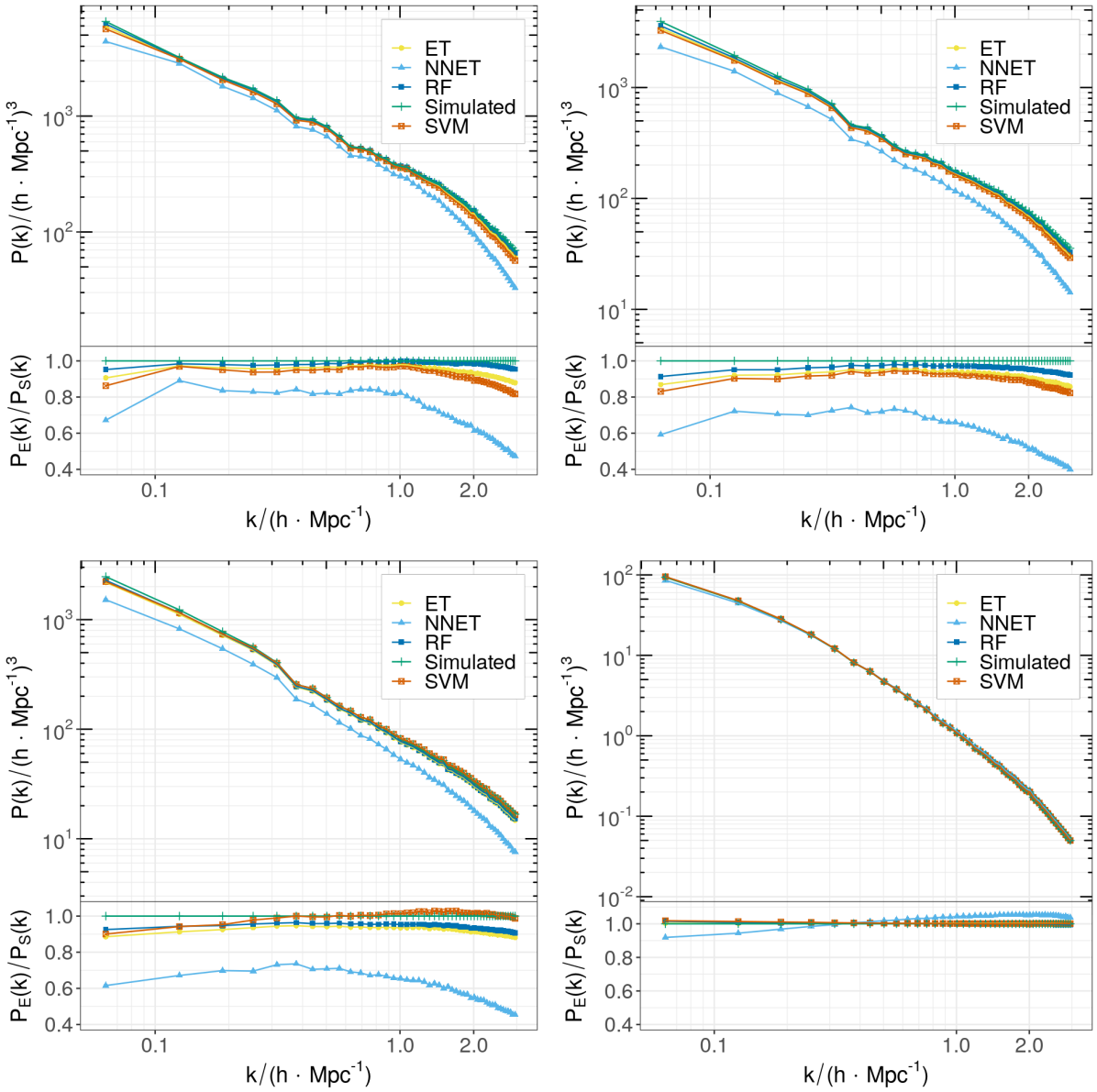

Figure A.1: The Power Spectra of the estimated Density Fields, at each redshifts and using the four algorithms with default hyper-parameters.. **Top Panel:** Power Spectrum Curves. **Bottom Panel**: Power Spectrum Ratio ($P_E(k)$ = Estimated;$P_S(k)$ = Simulated).

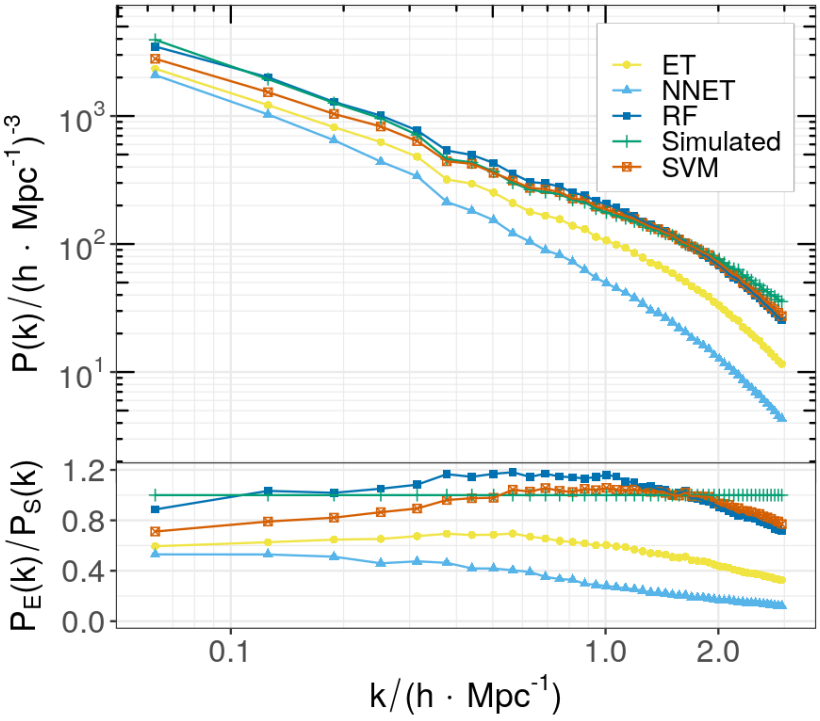

Figure A.2: The Power Spectra of the estimated Density Field using the four algorithms with default hyper-parameters on the 2D Dataset. **Top Panel:** Power Spectra Curves. **Bottom Panel**: Power Spectra Ratio ($P_E(k)$ = Estimated;$P_S(k)$ = Simulated).

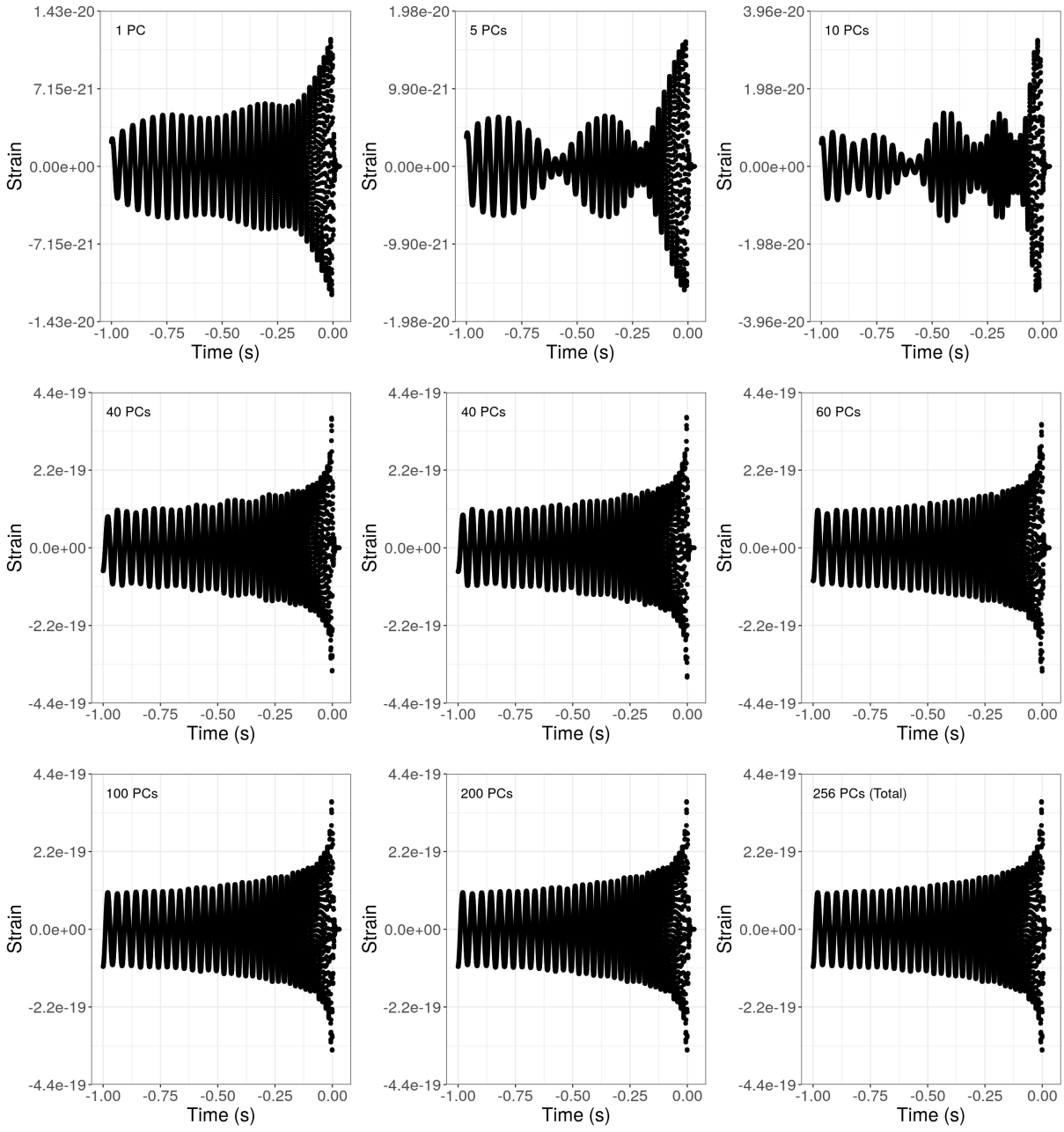

Figure A.3: Cumulative PC Reconstruction of a Time Domain Waveform for nine cases with increasing numbers of PCs.

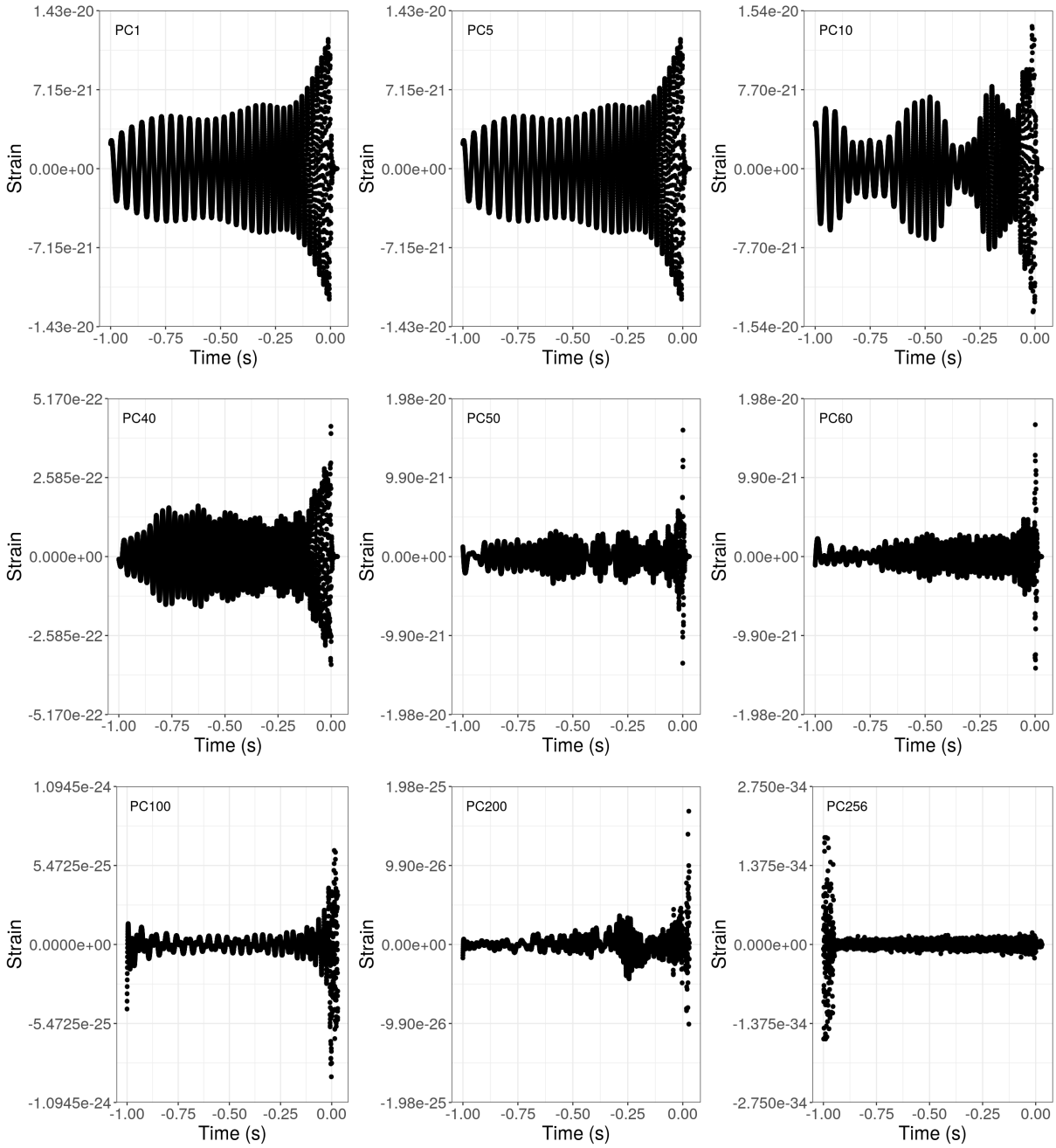

Figure A.4: Single PC Reconstruction of a Time Domain Waveform for nine cases with increasing PCs.

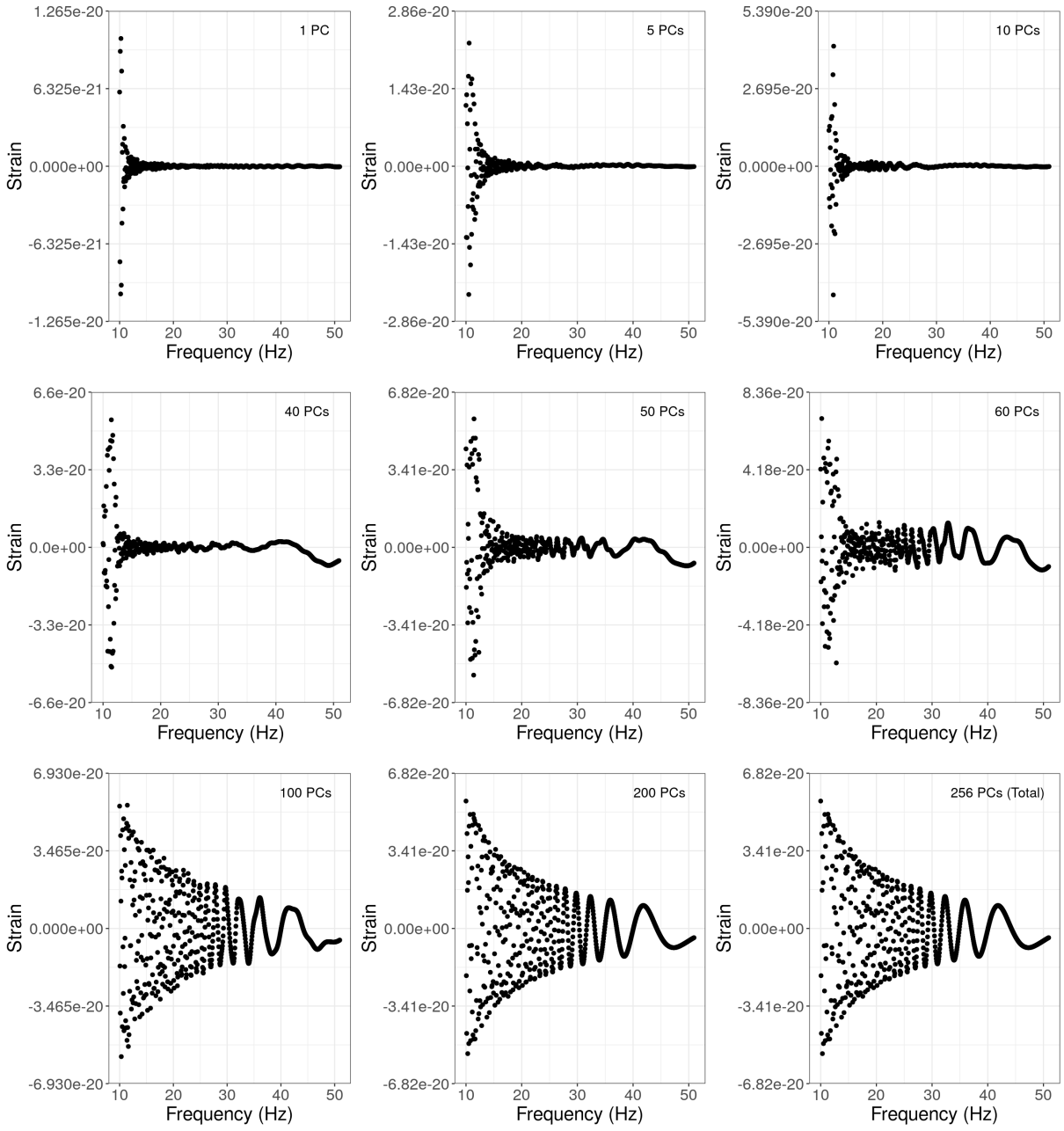

Figure A.5: Cumulative PC Reconstruction of a Frequency Domain Waveform for nine cases with increasing numbers of PCs.

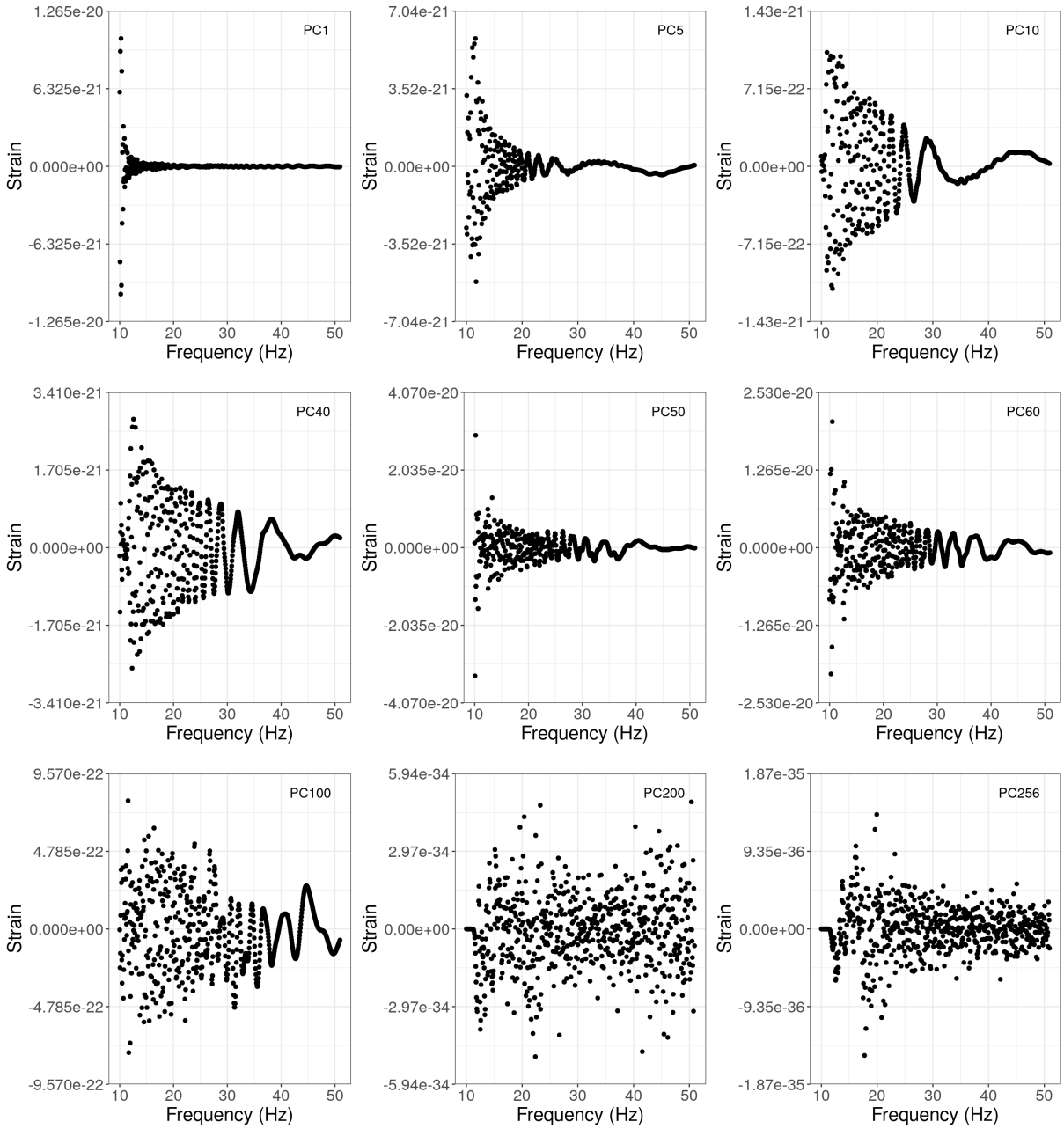

Figure A.6: Single PC Reconstruction of a Frequency Domain Waveform for nine cases with increasing PCs.

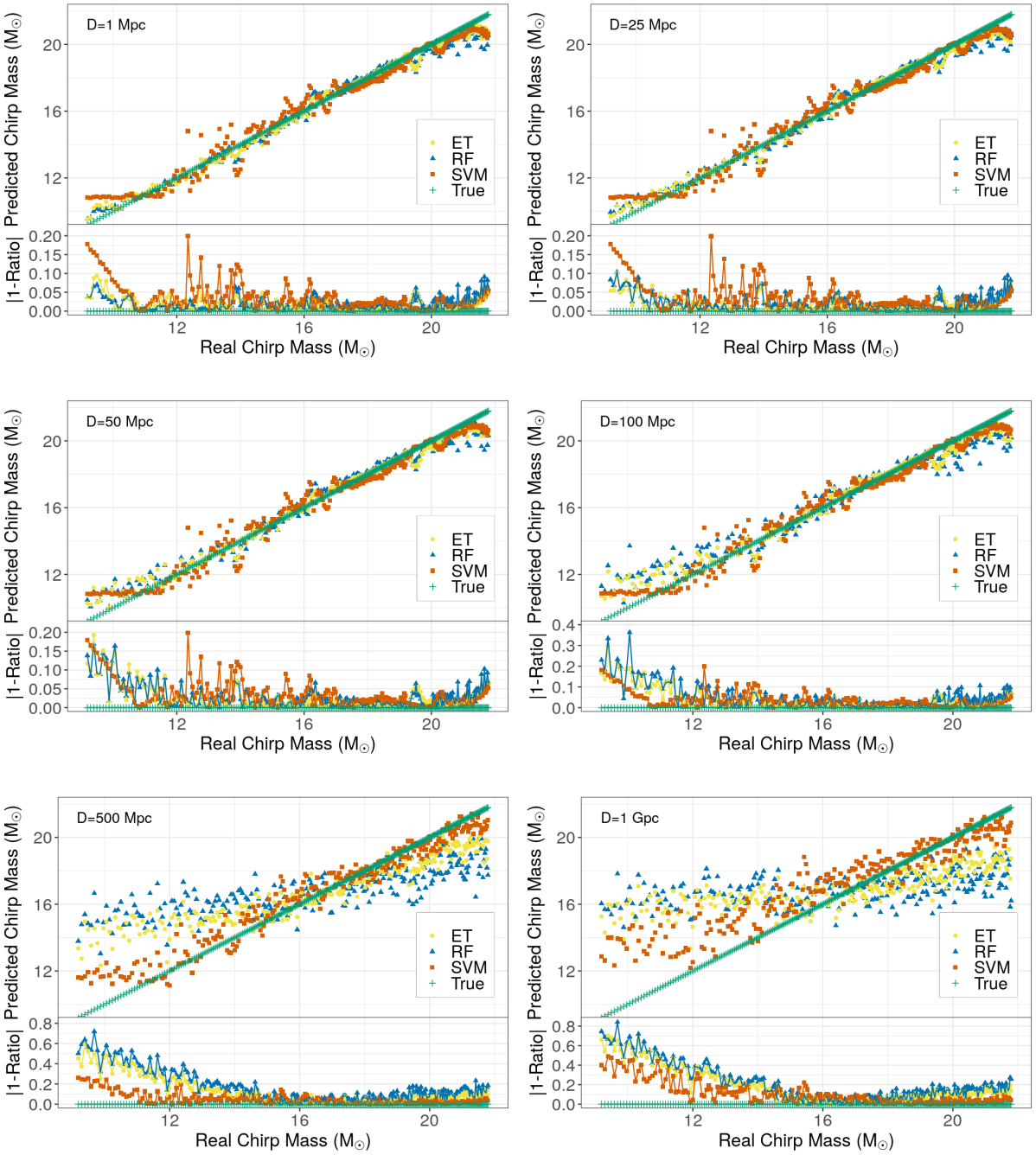

Figure A.7: Default Predictions vs Ground Truth in Time Domain with Unoptimized Algorithms and Optimized PCs.The Chirp Masses were estimated using the three algorithms, on waveforms concerning GW emmiting systems at increasing distances (decreasing signal-to-noise ratios). **Top Panel:** Estimated Chirp Mass vs Ground Truth. **Bottom Panel**: Chirp Mass Ratio $\left(\left|1 - \frac{Estimated}{Real}\right|\right)$. **Units:** Solar Masses ($M_\odot$).

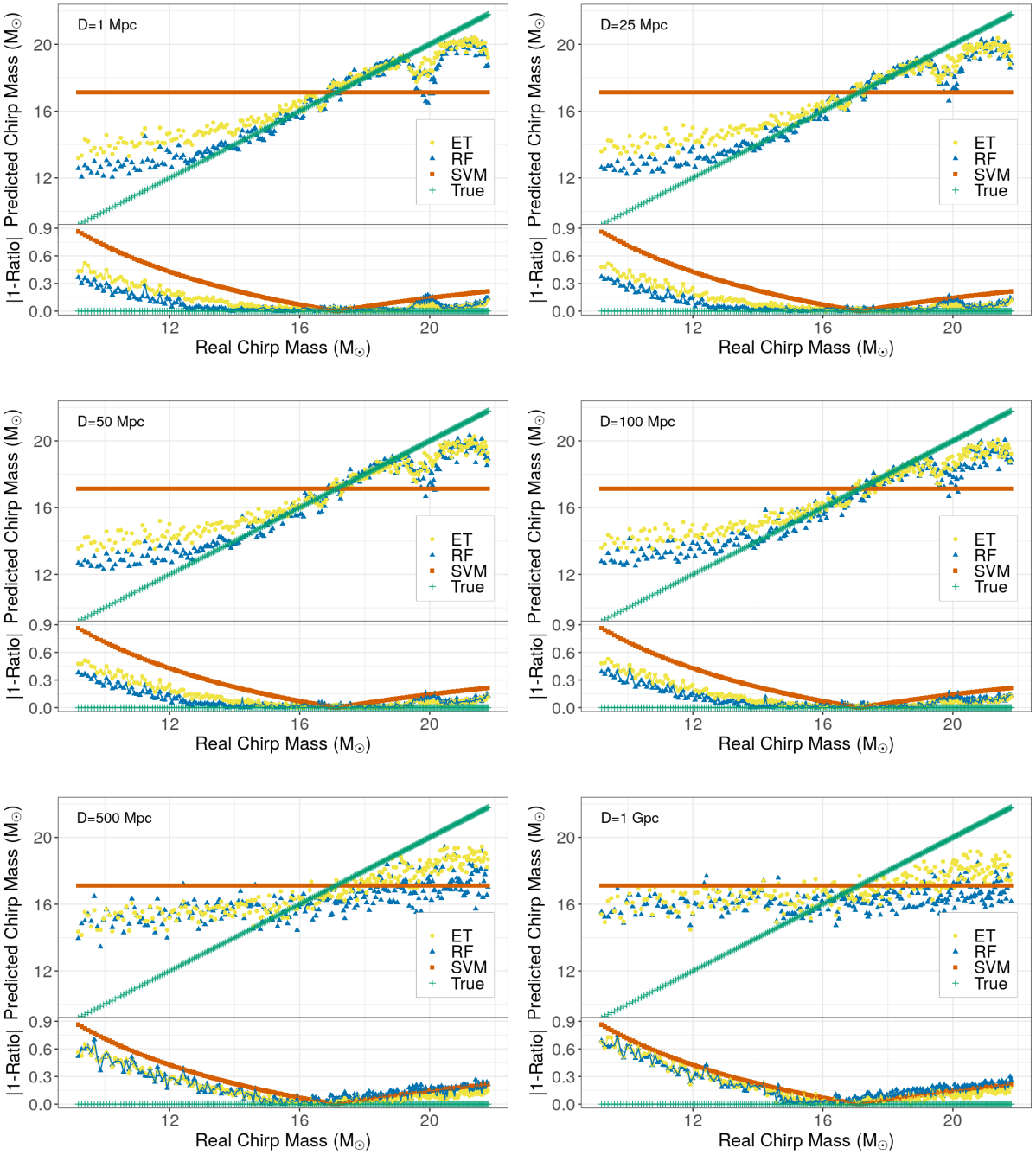

Figure A.8: Default Predictions vs Ground Truth in Time Domain with both Unoptimized Algorithms and PCs. The Chirp Masses were estimated using the three algorithms, on waveforms concerning GW emmiting systems at increasing distances (decreasing signal-to-noise ratios). **Top Panel:** Estimated Chirp Mass vs Ground Truth. **Bottom Panel**: Chirp Mass Ratio $\left(\left|1 - \frac{Estimated}{Real}\right|\right)$. **Units:** Solar Masses ($M_\odot$).

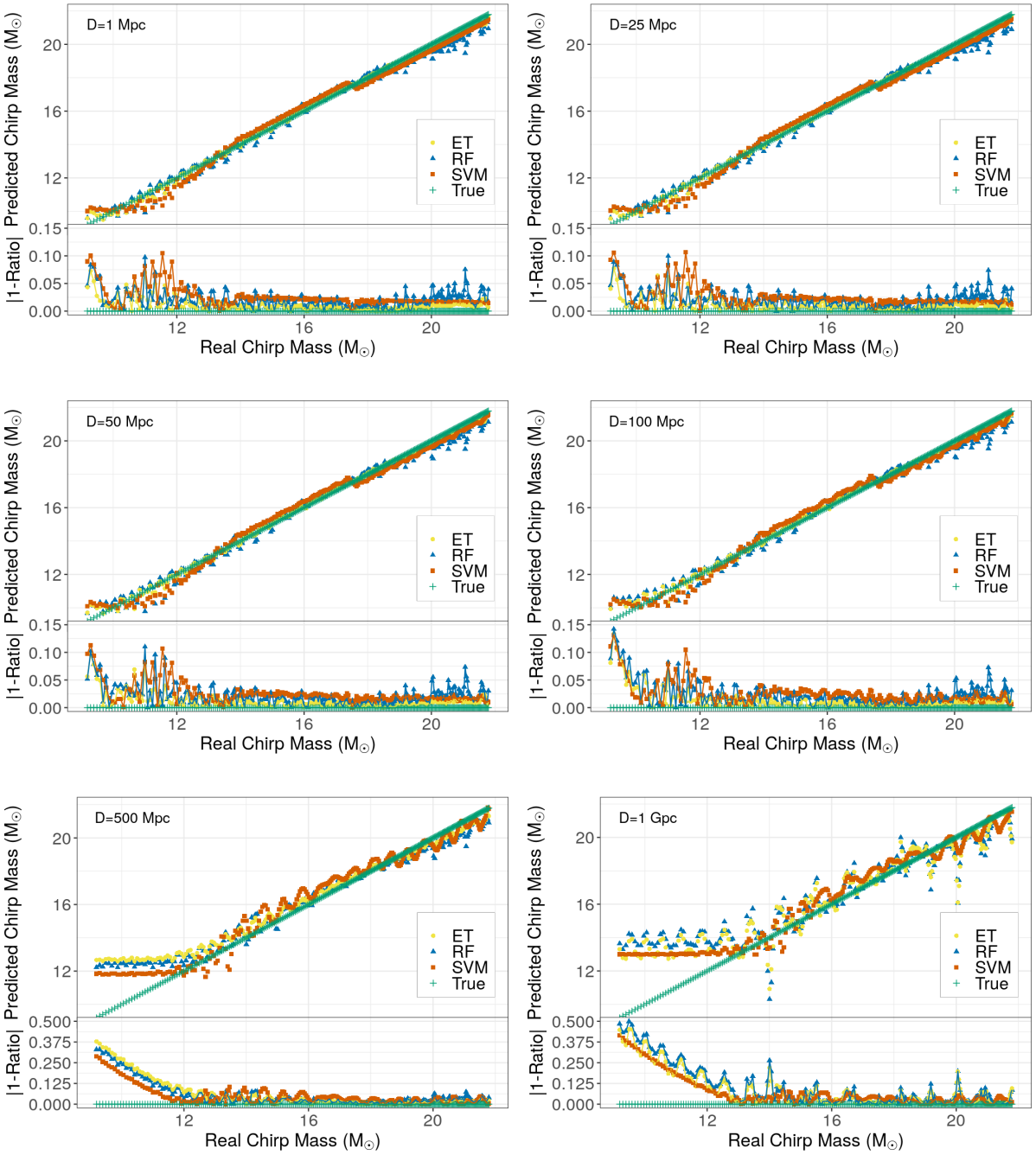

Figure A.9: Default Predictions vs Ground Truth in Frequency Domain with Unoptimized Algorithms and Optimized PCs. The Chirp Masses were estimated using the three algorithms, on waveforms concerning GW emmiting systems at increasing distances (decreasing signal-to-noise ratios). **Top Panel:** Estimated Chirp Mass vs Ground Truth. **Bottom Panel**: Chirp Mass Ratio $\left(\left|1 - \frac{Estimated}{Real}\right|\right)$. **Units:** Solar Masses ($M_\odot$).

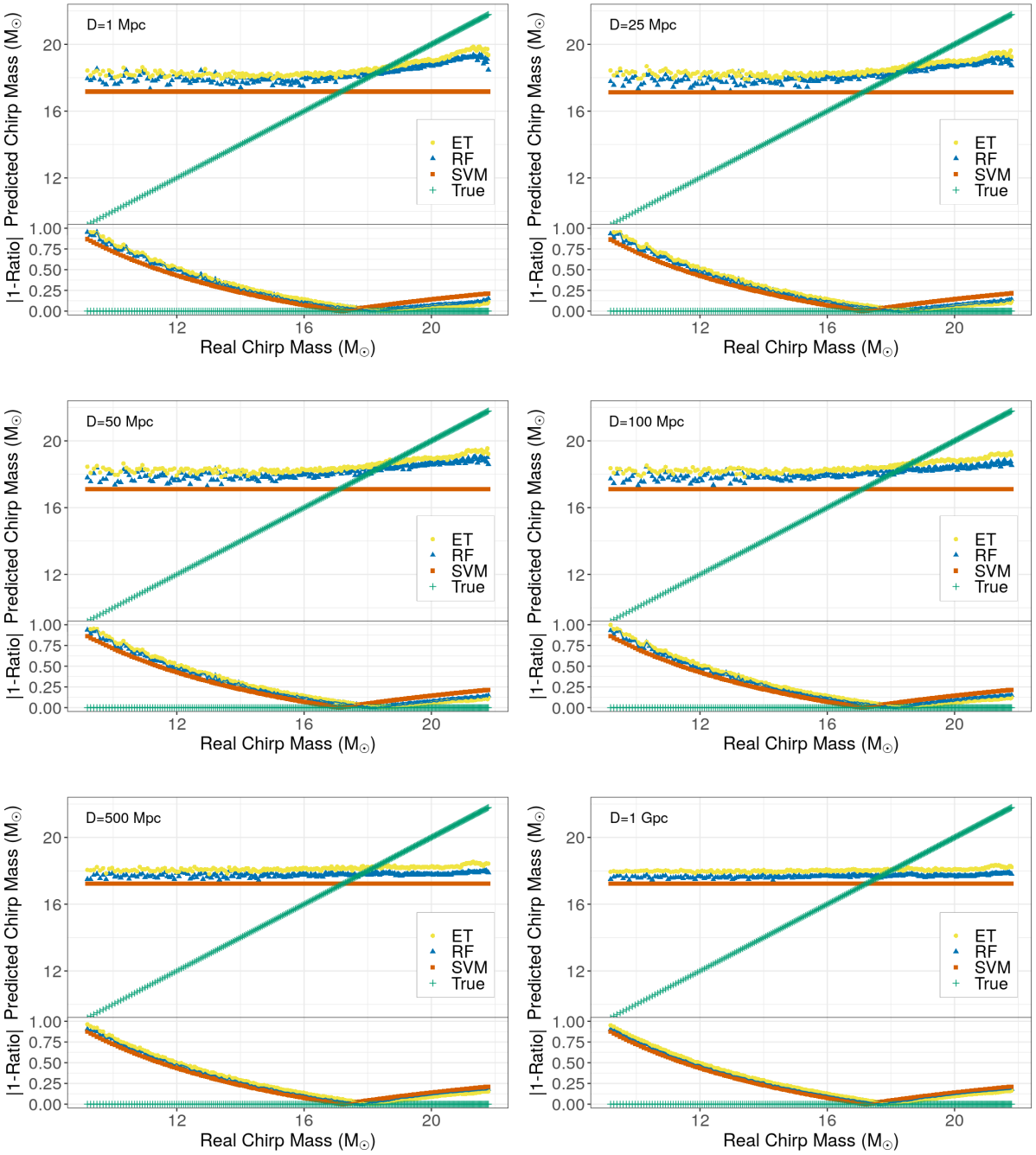

Figure A.10: Default Predictions vs Ground Truth in Frequency Domain with both Unoptimized Algorithms and PCs. The Chirp Masses were estimated using the three algorithms, on waveforms concerning GW emmiting systems at increasing distances (decreasing signal-to-noise ratios). **Top Panel:** Estimated Chirp Mass vs Ground Truth. **Bottom Panel**: Chirp Mass Ratio $\left(\left|1 - \frac{Estimated}{Real}\right|\right)$. **Units:** Solar Masses ($M_\odot$).

Bibliography

[1] T. Hey G. Bell and A. Szalay. “Beyond the Data Deluge”. In: *Science* 323.5919 (2009), pp.1297–1298. doi: 10.1126/science.1170411.

[2] LIGO Scientific Collaboration. *LIGO Algorithm Library - LALSuite*. free software (GPL). 2018. doi: 10.7935/GT1W-FZ16.

[3] Samantha A. Usman et al. “The PyCBC search for gravitational waves from compact binary coalescence”. In: *Classical and Quantum Gravity* 33.21, 215004 (Nov. 2016), p. 215004. doi: 10.1088/0264-9381/33/21/215004. arXiv: 1508.02357 [gr-qc].

[4] Tito Dal Canton et al. “Implementing a search for aligned-spin neutron star-black hole systems with advanced ground based gravitational wave detectors”. In: *Physical Review D* 90.8, 082004 (Oct. 2014), p. 082004. doi: 10.1103/PhysRevD.90.082004. arXiv: 1405.6731 [gr-qc].

[5] N. Aghanim et al. “Planck 2018 results”. In: *Astronomy & Astrophysics* 641 (Sept. 2020), A6. issn: 1432-0746. doi: 10.1051/0004-6361/201833910. url: http://dx.doi.org/10.1051/0004-6361/201833910.

[6] Robert M. Wald. *General Relativity*. University of Chicago Press, 1984.

[7] John A. Peacock. *Cosmological Physics*. Cambridge University Press, 2010.

[8] T. Padmanabhan. *Structure Formation in the Universe*. 1993.

[9] Andrew R. Liddle and David H. Lyth. *Cosmological Inflation and Large-Scale Structure*. Cambridge University Press, 2000. doi: 10.1017/CBO9781139175180.

[10] Volker Springel. “High Performance Computing and Numerical Modelling”. In: *Saas-Fee Course* 43 (Jan. 2016), p. 251. doi: 10.1007/978-3-662-47890-5_3. arXiv: 1412.5187 [astro-ph.GA].

[11] Edward W. Kolb and Michael S. Turner. *The early universe*. Vol. 69. 1990.

[12] Emiliano Sefusatti et al. “Cosmology and the bispectrum”. In: *Physical Review D* 74.2, 023522 (July 2006), p. 023522. doi: 10.1103/PhysRevD.74.023522. arXiv: astro-ph/0604505 [astro-ph].

[13] E. M. Lifshitz. “On the gravitational stability of the expanding universe”. In: *Zhurnal Eksperimentalnoi i Teoreticheskoi Fiziki* 16 (Jan. 1946), pp. 587–602.

[14] Y. B. Zel’Dovich. “Reprint of 1970A&A.....5...84Z. Gravitational instability: an approximate theory for large density perturbations.” In: *Astronomy & Astrophysics* 500 (Mar. 1970), pp. 13–18.

[15] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. 1981.

[16] Sverre J. Aarseth. *Gravitational N-Body Simulations: Tools and Algorithms*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, 2003. doi: 10.1017/CBO9780511535246.

[17] Josh Barnes and Piet Hut. “A hierarchical O(N log N) force-calculation algorithm”. In: *Nature* 324.6096 (Dec. 1986), pp. 446–449. doi: 10.1038/324446a0.

[18] R. Hockney, S. Goel, and J. Eastwood. “A 10000 particle molecular dynamics model with long range forces”. In: *Chemical Physics Letters* 21 (1973), pp. 589–591.

[19] H. M. P. Couchman, P. A. Thomas, and F. R. Pearce. “Hydra: an Adaptive-Mesh Implementation of P 3M-SPH”. In: *apj* 452 (Oct. 1995), p. 797. doi: 10.1086/176348. arXiv: astro-ph/9409058 [astro-ph].

[20] Antonio J. C. da Silva. “Hydrodynamic Simulations of the Zunyaev Zel’dovich Effect”. PhD thesis. University of Sussex, Sept. 2002.

[21] B. P. Abbott et al. “Observation of Gravitational Waves from a Binary Black Hole Merger”. In: *Phys. Rev. Lett.* 116 (6 Feb. 2016), p. 061102. doi: 10.1103/PhysRevLett.116.061102. url: https://link.aps.org/doi/10.1103/PhysRevLett.116.061102.

[22] Albert Einstein. “Näherungsweise Integration der Feldgleichungen der Gravitation”. In: *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften (Berlin* (Jan. 1916), pp. 688–696.

[23] Albert Einstein. “Über Gravitationswellen”. In: *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften (Berlin* (Jan. 1918), pp. 154–167.

[24] Chiara Caprini and Daniel G. Figueroa. “Cosmological backgrounds of gravitational waves”. In: *Classical and Quantum Gravity* 35.16, 163001 (Aug. 2018), p. 163001. doi: 10.1088/1361-6382/aac608. arXiv: 1801.04268 [astro-ph.CO].

[25] B. S. Sathyaprakash and B. F. Schutz. “Physics, Astrophysics and Cosmology with Gravitational Waves”. In: *Living Rev. Rel.* 12 (2009), p. 2. doi: 10.12942/lrr-2009-2. arXiv: 0903.0338 [gr-qc].

[26] Sascha Husa. “Michele Maggiore: Gravitational waves. Volume 1: theory and experiments. Oxford University Press, 2007, 576p., GBP47.00, ISBN13: 978-0-19-857074-5”. In: *General Relativity and Gravitation* 41.7 (July 2009), pp. 1667–1669. doi: 10.1007/s10714-009-0762-5.

[27] Éanna É. Flanagan and Scott A. Hughes. “The basics of gravitational wave theory”. In: *New Journal of Physics* 7.1 (Sept. 2005), p. 204. doi: 10.1088/1367-2630/7/1/204. arXiv: gr-qc/0501041 [astro-ph].

[28] Luc Blanchet. “Gravitational radiation from post-Newtonian sources and inspiralling compact binaries”. In: *Living Rev. Rel.* 9 (2006), p. 4.

[29] Alejandro Bohé et al. “Improved effective-one-body model of spinning, nonprecessing binary black holes for the era of gravitational-wave astrophysics with advanced detectors”. In: *Physical Review D* 95.4, 044028 (Feb. 2017), p. 044028. doi: 10.1103/PhysRevD.95.044028. arXiv: 1611.03703 [gr-qc].

[30] B. S. Sathyaprakash and S. V. Dhurandhar. “Choice of filters for the detection of gravitational waves from coalescing binaries”. In: *Phys. Rev. D* 44 (12 Dec. 1991), pp. 3819–3834. doi: 10.1103/PhysRevD.44.3819. url: https://link.aps.org/doi/10.1103/PhysRevD.44.3819.

[31] Mischa Knabenhans et al. “Euclid preparation: II. The EUCLIDEMULATOR - a tool to compute the cosmology dependence of the nonlinear matter power spectrum”. In: *MNRAS* 484.4 (Apr. 2019), pp. 5509–5529. doi: 10.1093/mnras/stz197. arXiv: 1809.04695 [astro-ph.CO].

[32] Siamak Ravanbakhsh et al. “Estimating Cosmological Parameters from the Dark Matter Distribution”. In: *arXiv e-prints*, arXiv:1711.02033 (Nov. 2017), arXiv:1711.02033. arXiv: 1711.02033 [astro-ph.CO].

[33] Siyu He et al. “Learning to predict the cosmological structure formation”. In: *Proceedings of the National Academy of Science* 116.28 (June 2019), pp. 13825–13832. doi: 10.1073/pnas.1821458116. arXiv: 1811.06533 [astro-ph.CO].

[34] Daniel George and E. A. Huerta. “Deep Learning for real-time gravitational wave detection and parameter estimation: Results with Advanced LIGO data”. In: *Physics Letters B* 778 (Mar. 2018), pp. 64–70. doi: 10.1016/j.physletb.2017.12.053. arXiv: 1711.03121 [gr-qc].

[35] Alvin J. K. Chua, Chad R. Galley, and Michele Vallisneri. “Reduced-Order Modeling with Artificial Neurons for Gravitational-Wave Inference”. In: *Phys. Rev. Lett.* 122 (21 May 2019), p. 211101. doi: 10.1103/PhysRevLett.122.211101. url: https://link.aps.org/doi/10.1103/PhysRevLett.122.211101.

[36] Alvin J. K. Chua and Michele Vallisneri. “Learning Bayesian Posteriors with Neural Networks for Gravitational-Wave Inference”. In: *Physical Review Letters* 124.4, 041102 (Jan. 2020), p. 041102. doi: 10.1103/PhysRevLett.124.041102. arXiv: 1909.05966 [gr-qc].

[37] Leo Breiman. “Machine Learning, Volume 45, Number 1 - SpringerLink”. In: *Machine Learning* 45 (Oct. 2001), pp. 5–32. doi: 10.1023/A:1010933404324.

[38] J. R. Quinlan. “Induction of Decision Trees”. In: *Mach. Learn.* 1.1 (Mar. 1986), pp. 81–106. issn: 0885-6125. doi: 10.1023/A:1022643204877. url: https://doi.org/10.1023/A:1022643204877.

[39] Pierre Geurts, Damien Ernst, and Louis Wehenkel. “Extremely Randomized Trees”. In: *Machine Learning* 63 (Apr. 2006), pp. 3–42. doi: 10.1007/s10994-006-6226-1.

[40] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. issn: 1573-0565. doi: 10.1023/A:1022627411411. url: https://doi.org/10.1023/A:1022627411411.

[41] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. 4th. USA: Academic Press, Inc., 2008. isbn: 1597492728.

[42] Evgenia Dimitriadou et al. “E1071: Misc Functions of the Department of Statistics (E1071), TU Wien”. In: vol. 1. Nov. 2009.

[43] S. C. Kleene. “Representation of events in nerve nets and finite automata”. In: *Automata Studies*. Ed. by Claude Shannon and John McCarthy. Princeton, NJ: Princeton University Press, 1956, pp. 3–41.

[44] Augustin-Louis Cauchy. “ANALYSE MATHÉMATIQUE. – Méthodc générale pour la résolution des systèmes d’équations simultanées”. In: 2009.

[45] Henry J Kelley. “Gradient theory of optimal flight paths”. In: *Ars Journal* 30.10 (1960), pp. 947–954.

[46] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[47] Karl Pearson F.R.S. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 02.11 (1901), pp. 559–572. doi: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. url: https://doi.org/10.1080/14786440109462720.

[48] Gareth James et al. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. url: https://faculty.marshall.usc.edu/gareth-james/ISL/.

[49] Ulf Grenander. "Stochastic processes and statistical inference". In: *Ark. Mat.* 1.3 (Oct. 1950), pp. 195–277. doi: 10.1007/BF02590638. url: https://doi.org/10.1007/BF02590638.

[50] John A. Rice and B. W. Silverman. "Estimating the Mean and Covariance Structure Nonparametrically When the Data are Curves". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 53.1 (1991), pp. 233–243. issn: 00359246. url: http://www.jstor.org/stable/2345738.

[51] Bernard W. Silverman. "Smoothed functional principal components analysis by choice of norm". In: *Ann. Statist.* 24.1 (Feb. 1996), pp. 1–24. doi: 10.1214/aos/1033066196. url: https://doi.org/10.1214/aos/1033066196.

[52] Leo Breiman. *Bagging Predictors*. 1994.

[53] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2020. url: https://www.R-project.org/.

[54] Stéphane Colombi et al. "Accurate estimators of power spectra in N-body simulations". In: *MNRAS* 393.2 (Feb. 2009), pp. 511–526. doi: 10.1111/j.1365-2966.2008.14176.x. arXiv: 0811.0313 [astro-ph].

[55] J. M. Bardeen et al. "The Statistics of Peaks of Gaussian Random Fields". In: *apj* 304 (May 1986), p. 15. doi: 10.1086/164143.

[56] Naoshi Sugiyama. "Cosmic Background Anisotropies in Cold Dark Matter Cosmology". In: *apjs* 100 (Oct. 1995), p. 281. doi: 10.1086/192220. arXiv: astro-ph/9412025 [astro-ph].

[57] Planck Collaboration et al. "Planck 2015 results. XIII. Cosmological parameters". In: *Astronomy & Astrophysics* 594, A13 (Oct. 2016), A13. doi: 10.1051/0004-6361/201525830. arXiv: 1502.01589 [astro-ph.CO].

[58] L. B. Lucy. "A numerical approach to the testing of the fission hypothesis." In: *Astronomical Journal* 82 (Dec. 1977), pp. 1013–1024. doi: 10.1086/112164.

[59] Karline Soetaert. *plot3D: Plotting Multi-Dimensional Data*. R package version 1.3. 2019. url: https://CRAN.R-project.org/package=plot3D.

[60] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer, 2002. url: http://www.stats.ox.ac.uk/pub/MASS4.

[61] Andy Liaw and Matthew Wiener. "Classification and Regression by randomForest". In: *R News* 2.3 (2002), pp. 18–22. url: https://CRAN.R-project.org/doc/Rnews/.

[62] Jaak Simm, Ildefons Magrans de Abril, and Masashi Sugiyama. *Tree-Based Ensemble Multi-Task Learning Method for Classification and Regression*. 6. The Institute of Electronics, Information and Communication Engineers, 2014, pp. 1677–1681. url: http://CRAN.R-project.org/package=extraTrees.

[63] Max Kuhn. "Building Predictive Models in R Using the caret Package". In: *Journal of Statistical Software, Articles* 28.5 (2008), pp. 1–26. issn: 1548-7660. doi: 10.18637/jss.v028.i05. url: https://www.jstatsoft.org/v028/i05.

[64] J. O. Ramsay, Spencer Graves, and Giles Hooker. *fda: Functional Data Analysis*. R package version 5.1.5.1. 2020. url: https://CRAN.R-project.org/package=fda.

[65] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. isbn: 978-3-319-24277-4. url: https://ggplot2.tidyverse.org.

[66] David Wolpert. "Stacked Generalization". In: *Neural Networks* 5 (Dec. 1992), pp. 241–259. doi: 10.1016/S0893-6080(05)80023-1.

[67] Xiaoying Xu et al. "A First Look at Creating Mock Catalogs with Machine Learning Techniques". In: *The Astrophysical Journal* 772.2, 147 (Aug. 2013), p. 147. doi: 10.1088/0004-637X/772/2/147. arXiv: 1303.1055 [astro-ph.CO].

[68] Luisa Lucie-Smith et al. "Machine learning cosmological structure formation". In: *MNRAS* 479.3 (Sept. 2018), pp. 3405–3414. doi: 10.1093/mnras/sty1719. arXiv: 1802.04271 [astro-ph.CO].

[69] Shankar Agarwal et al. "PkANN - I. Non-linear matter power spectrum interpolation through artificial neural networks". In: *MNRAS* 424.2 (Aug. 2012), pp. 1409–1418. doi: 10.1111/j.1365-2966.2012.21326.x. arXiv: 1203.1695 [astro-ph.CO].

[70] Shankar Agarwal et al. "PkANN - II. A non-linear matter power spectrum interpolator developed using artificial neural networks". In: *MNRAS* 439.2 (Apr. 2014), pp. 2102–2121. doi: 10.1093/mnras/stu090. arXiv: 1312.2101 [astro-ph.CO].

[71] Siamak Ravanbakhsh et al. "Estimating Cosmological Parameters from the Dark Matter Distribution". In: *arXiv e-prints*, arXiv:1711.02033 (Nov. 2017), arXiv:1711.02033. arXiv: 1711.02033 [astro-ph.CO].

[72] Amrita Mathuriya et al. "CosmoFlow: Using Deep Learning to Learn the Universe at Scale". In: *arXiv e-prints*, arXiv:1808.04728 (Aug. 2018), arXiv:1808.04728. arXiv: 1808.04728 [astro-ph.CO].

[73] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. url: http://tensorflow.org/.

[74] Andres C. Rodríguez et al. "Fast cosmic web simulations with generative adversarial networks". In: *Computational Astrophysics and Cosmology* 5.1, 4 (Nov. 2018), p. 4. doi: 10.1186/s40668-018-0026-4. arXiv: 1801.09070 [astro-ph.CO].

[75] Nathanaël Perraudin et al. "Cosmological N-body simulations: a challenge for scalable generative models". In: *Computational Astrophysics and Cosmology* 6.1, 5 (Dec. 2019), p. 5. doi: 10.1186/s40668-019-0032-1. arXiv: 1908.05519 [physics.comp-ph].

[76] Marion Ullmo, Aurélien Decelle, and Nabila Aghanim. "Encoding large scale cosmological structure with Generative Adversarial Networks". In: *arXiv e-prints*, arXiv:2011.05244 (Nov. 2020), arXiv:2011.05244. arXiv: 2011.05244 [astro-ph.CO].

[77] Siyu He et al. "Learning to predict the cosmological structure formation". In: *Proceedings of the National Academy of Science* 116.28 (July 2019), pp. 13825–13832. doi: 10.1073/pnas.1821458116. arXiv: 1811.06533 [astro-ph.CO].

[78] Yu Feng et al. "FASTPM: a new scheme for fast simulations of dark matter and haloes". In: *MNRAS* 463.3 (Dec. 2016), pp. 2273–2286. doi: 10.1093/mnras/stw2123. arXiv: 1603.00476 [astro-ph.CO].

[79] Renan Alves de Oliveira et al. "Fast and Accurate Non-Linear Predictions of Universes with Deep Learning". In: *arXiv e-prints*, arXiv:2012.00240 (Nov. 2020), arXiv:2012.00240. arXiv: 2012.00240 [astro-ph.CO].

[80] Alan F. Heavens, Raul Jimenez, and Ofer Lahav. “Massive lossless data compression and multiple parameter estimation from galaxy spectra”. In: *MNRAS* 317.4 (Oct. 2000), pp. 965–972. doi: 10.1046/j.1365-8711.2000.03692.x. arXiv: astro-ph/9911102 [astro-ph].

[81] Alan F. Heavens et al. “Massive data compression for parameter-dependent covariance matrices”. In: *MNRAS* 472.4 (Dec. 2017), pp. 4244–4250. doi: 10.1093/mnras/stx2326. arXiv: 1707.06529 [astro-ph.CO].

[82] F. Barone et al. “A Neural Network-based ARX Model of Virgo Noise”. In: *arXiv e-prints*, astro-ph/9906107 (June 1999), astro–ph/9906107. arXiv: astro-ph/9906107 [astro-ph].

[83] Rahul Biswas et al. “Application of machine learning algorithms to the study of noise artifacts in gravitational-wave data”. In: *Physical Review D* 88.6, 062003 (Sept. 2013), p. 062003. doi: 10.1103/PhysRevD.88.062003. arXiv: 1303.6984 [astro-ph.IM].

[84] Salvatore Rampone et al. “Neural Network Aided Glitch-Burst Discrimination and Glitch Classification”. In: *International Journal of Modern Physics C* 24.11, 1350084 (Nov. 2013), p. 1350084. doi: 10.1142/S0129183113500848. arXiv: 1401.5941 [astro-ph.IM].

[85] Thomas S. Adams et al. “Gravitational-wave detection using multivariate analysis”. In: *Physical Review D* 88.6, 062006 (Sept. 2013), p. 062006. doi: 10.1103/PhysRevD.88.062006. arXiv: 1305.5714 [gr-qc].

[86] Kyungmin Kim et al. “Application of artificial neural network to search for gravitational-wave signals associated with short gamma-ray bursts”. In: *Classical and Quantum Gravity* 32.24, 245002 (Dec. 2015), p. 245002. doi: 10.1088/0264-9381/32/24/245002. arXiv: 1410.6878 [astro-ph.IM].

[87] Daniel George and E. A. Huerta. “Deep neural networks to enable real-time multimessenger astrophysics”. In: *Physical Review D* 97.4, 044039 (Feb. 2018), p. 044039. doi: 10.1103/PhysRevD.97.044039. arXiv: 1701.00008 [astro-ph.IM].

[88] Daniel George and E. A. Huerta. “Deep Learning for real-time gravitational wave detection and parameter estimation: Results with Advanced LIGO data”. In: *Physics Letters B* 778 (Mar. 2018), pp. 64–70. doi: 10.1016/j.physletb.2017.12.053. arXiv: 1711.03121 [gr-qc].

[89] Alvin J. K. Chua and Michele Vallisneri. “Learning Bayesian Posteriors with Neural Networks for Gravitational-Wave Inference”. In: *Physical Review Letters* 124.4, 041102 (Jan. 2020), p. 041102. doi: 10.1103/PhysRevLett.124.041102. arXiv: 1909.05966 [gr-qc].

[90] Hunter Gabbard et al. “Bayesian parameter estimation using conditional variational autoencoders for gravitational-wave astronomy”. In: (Sept. 2019). arXiv: 1909.06296 [astro-ph.IM].

[91] Stephen R. Green, Christine Simpson, and Jonathan Gair. “Gravitational-wave parameter estimation with autoregressive neural network flows”. In: *Physical Review D* 102.10, 104057 (Nov. 2020), p. 104057. doi: 10.1103/PhysRevD.102.104057. arXiv: 2002.07656 [astro-ph.IM].

[92] Stephen R. Green and Jonathan Gair. “Complete parameter inference for GW150914 using deep learning”. In: (Aug. 2020). arXiv: 2008.03312 [astro-ph.IM].

[93] Joongoo Lee et al. “Deep Learning Model on Gravitational Waveforms in Merging and Ringdown Phases of Binary Black Hole Coalescences”. In: *arXiv e-prints*, arXiv:2101.05685 (Jan. 2021), arXiv:2101.05685. arXiv: 2101.05685 [astro-ph.IM].